



# The Design and Development of An Information Retrieval System for the EAMATE Data

**Natalie Willman  
Laura L. Downey**

U.S. DEPARTMENT OF COMMERCE  
Technology Administration  
National Institute of Standards  
and Technology  
Gaithersburg, MD 20899

*Welcome*

TO THE EAMATE PROTOTYPE



DESIGNED AND DEVELOPED BY

**Natalie E. Willman,  
Senior Computer Scientist**

**Laura L. Downey,  
Computer Scientist**

**National Institute of Standards  
and Technology**

Continue

QC  
100  
.U56  
NO.5394  
1994

**NIST**



# **The Design and Development of An Information Retrieval System for the EAMATE Data**

**Natalie Willman  
Laura L. Downey**

U.S. DEPARTMENT OF COMMERCE  
Technology Administration  
National Institute of Standards  
and Technology  
Gaithersburg, MD 20899

April 1994



**U.S. DEPARTMENT OF COMMERCE  
Ronald H. Brown, Secretary**

**TECHNOLOGY ADMINISTRATION  
Mary L. Good, Under Secretary for Technology**

**NATIONAL INSTITUTE OF STANDARDS  
AND TECHNOLOGY  
Arati Prabhakar, Director**





## **Acknowledgments**

Many people, in addition to the authors of this report, have contributed to the program outlined in this publication, and the authors are most appreciative of that assistance.

The support and information provided by the project team at the Social Security Administration (SSA) was invaluable during the course of the project. Thanks are due to the following individuals from SSA: Malcolm Ewell, Janice Whelchel, Ruth Ann Clem, and Anita Cohen. In addition, thanks are due to the Executive Staff at SSA for their support and encouragement.

Thanks are due to Donna Harman (National Institute of Standards and Technology), Peter Willett (University of Sheffield), and Sandy Robertson (University of Sheffield) for many helpful ideas and comments in the development of the search engine.

This work has been funded by the Social Security Administration, Department of Health and Human Services.



# Table of Contents

## Volume I - Report

1. Introduction and Description of Problem .....	11
2. Prototype System Components.....	15
2.1 Hardware Components.....	15
2.2 Software Components.....	15
3. System and Software Design .....	17
3.1 High Level Design .....	17
3.2 Prototype Observations .....	18
3.3 EAMATE Record Extraction Tool .....	20
3.4 High Level Data Constructs.....	21
4. User Interface Design and Implementation .....	27
4.1 Reasons for Development.....	27
4.2 Overview of the EAMATE User Interface .....	27
4.3 Development.....	28
4.4 General Design and Screen Characteristics .....	28
4.5 Detailed Screen Descriptions .....	30
4.6 User Interface Software Organization.....	44
4.7 Phase Transition in the User Interface .....	52
4.8 Re-Engineering the Process .....	55
4.9 Quality Assurance Measures.....	56
5. Design of the "Best-Match" Search Algorithms .....	57
5.1 Reasons for Development.....	57
5.2 Previous Research in Best-Match Searching .....	57
5.3 Technical Description of Algorithms.....	58
5.4 Laboratory Results and Performance Evaluation.....	61
5.5 Areas of Further Research.....	66
6. Search Engine Design and Implementation.....	67
6.1 High Level Data Structures.....	67
6.2 Tunable Parameters.....	69
6.3 Source Code Modules .....	72
7. Conversion and Indexing of the EAMATE Data.....	77
7.1 Directory and File Structure Required.....	77
7.2 Walkthru of the Data Conversion Process .....	80
7.3 Walkthru of the Indexing Process.....	84
7.4 Overview of the Verification Programs .....	87

8. Searching of the EAMATE Data .....	91
8.1 Get Employer Header -- All Headers for the Given Year and EIN ...	91
8.2 Get Employer Header -- By Sequence Number .....	92
8.3 Single Query Request .....	92
8.4 Blanket Request .....	93
8.5 Get Employee Detail .....	94
8.6 Browse Report .....	94
8.7 Print Report.....	94
8.8 Add Matches .....	95
9. Communication Issues .....	97
10. Usability .....	99
10.1 Incorporating the Usability Engineering Lifecycle.....	99
10.2 Heuristic Evaluation as a Usability Inspection Method.....	101
10.3 Application Performance Measurements and Usage Statistics .....	104
10.4 User Satisfaction .....	109
10.5 Conclusions and Future Plans.....	110
11. References .....	111

## **Volume II - Appendices**

A. User Instructions .....	A-1
A.1 Step-By-Step Instructions .....	A-1
A.2 Data Entry Rules and Field Attributes .....	A-10
A.3 Visual Manual.....	A-12
B. Installation and Configuration .....	B-1
B.1 Client Workstation.....	B-1
B.2 File Server.....	B-5
C. System Error Messages.....	C-1
C.1 User Interface Error Messages .....	C-1
C.2 Search Engine Error Messages .....	C-20
D. Listing of the Code.....	D-1
D.1 User Interface Code .....	D-3
D.2 Search Engine Code.....	D-317
E. Scout Comments from EAMATE Testing Session .....	E-1
F. Preliminary Employer Report Statistics.....	F-1
F.1 Einstats File.....	F-1
F.2 Indexstats File .....	F-7

## Table of Figures

2-1	EAMATE Prototype System Configuration .....	16
4-1	Structure Diagram.....	29
4-2	Main Screen .....	31
4-3	Single Query Prompt Screen.....	32
4-4	Report Statistics Screen .....	33
4-5	Browse Report Prompt Screen.....	34
4-6	Print Report Prompt Screen .....	35
4-7	Exit Program Prompt Screen .....	36
4-8	Query Information Screen.....	37
4-9	Query Parameter Screen.....	38
4-10	Browse Report Information Screen.....	39
4-11	Potential Blanket Information Screen.....	40
4-12	Employee Detail Screen.....	41
4-13	Employer Detail Screen .....	42
4-14	Totals Detail Screen.....	42
4-15	Example Error Screen, Message Screen and Current Process Screen .....	43
4-16	Phase Transition Diagram.....	52
5-1	Format of Index and Instance Files.....	59
5-2	Structure of Hash Table .....	61

NOTE: Per the Social Security Administration's request, all social security numbers have been removed from any figure to ensure data protection.

**Table of Tables**

5-1 N-gram Experiment Results..... 63

5-2 Index Time and Size (Elapsed Time)..... 64

5-3 Scaling Experiments ..... 64

5-4 Pruning by Dice Coefficient Results..... 65

9-1 Listing of RPC Functions ..... 97

A-1 Data Entry Fields and Their Attributes..... A-11



## Table of Acronyms

AWR	Employer Reports of Wages which are filed on Paper
COM	Computer-Output to Microfilm
EAMATE	Employer Reports of Wages which are filed Electronically
EIN	Employer Identification Number
FICA	Federal Insurance Contributions Act
MRN	Microfilm Reference Number
NIST	National Institute of Standards and Technology
RPC	Remote Procedure Call
SSA	Social Security Administration
SSN	Social Security Number



## **1 Introduction and Description of Problem**

The Social Security Administration maintains records of wages earned by every person who has a social security number. These wages can be classified as self-employed wages, which are received electronically from the Internal Revenue Service (IRS), or as employee wages, which are received in reports from employers. The employer reports are filed in two different manners: electronically by large employers (EAMATE reports), and on paper by small employers (AWR reports). Large employers range in size from 100 employees to over 400,000 employees, and total approximately 175 million records per year. Small employers are those with less than 100 employees, and total approximately 67 million records per year.

Currently, all reports are maintained on microfilm. It has been estimated that the microfilm library at the Office of Central Records Operations (OCRO) in Baltimore, Maryland, contains 800 million feet, or 150,000 miles, of microfilm - - enough film to circle the globe six times.

When a person's wage record must be verified, it is often necessary to search the employer reports for this person's wages. The process is cumbersome as the film is difficult to read, and often employees are listed in no order, requiring a sequential search through the entire report. In addition, the particular reel of film needed is frequently damaged or already charged out to another person.

The high storage density and cost-effectiveness of optical disk technology suggested possibilities of automating these files. The large employer reports for 1991 and later years were available in electronic format, and the small employer reports are now being digitally captured and converted to text via Optical Character Recognition (OCR). A project was started to explore the feasibility of automating these files, and to re-engineer the scouting process for earnings verification in order to take advantage of an automated system.

When the project began, it was discovered that there were many problems with these reports which made the automation of these files much more difficult. These problems included such things as: variations in the manner of reporting personal names (ordering of the first and last names, initials, middle names, random punctuation and titles), misspellings in names, incorrect social security numbers, the necessity to deal with parts of hyphenated names, nicknames, and to resolve married and maiden names. In addition, an employer could file more than one report for any given year. It was important that the scout be able to search all the reports as one when looking for an employee, but that the reports maintain their inherent structure and could be viewed separately.

Therefore, the goals of the project were defined as follows:

1. To prove (or disprove) that an information retrieval system will benefit OCRO's earnings reference operation.
2. To develop a new indexing and searching methodology which has the power and flexibility to overcome inconsistencies in the data.
3. To design a highly functional user interface which provides communication between the search-engine and the user, while providing an intuitive and user-friendly environment.
4. To re-engineer the scouting process in order to empower the users to take full advantage of an automated system.
5. To identify potential problems in automation before moving to a full implementation.
6. To capture statistical data useful in the procurement of a full scale system.

The actual products of the project are a series of prototype systems to demonstrate the feasibility of automating the report files. These prototype systems explore high risk areas of automation, identify potential problems in automation, and demonstrate re-engineering concepts. In addition, the prototypes provide a vehicle for gathering statistical data and allow the users to provide input into the final system. The prototypes emphasize user-centered design concepts as well as the task analysis involved in the re-engineering effort.

The EAMATE prototype is the second of these systems. After this system, a prototype system will be built for the AWR and Suspense data sets. (The Suspense data set is a subset of records from the EAMATE and AWR data sets which contain known reporting errors in either the name or social security number). As these systems are built, they will be integrated to form a comprehensive retrieval system. The focus of this document is the evolution and design of the EAMATE prototype system, specifically the steps taken to achieve goals 2,3 and 4 from the list above.

In order to migrate the prototypes into a Request For Purchase (RFP) for a final production system, an expanded prototype will be built and maintained at SSA in OCRO. This system will begin with using the 1991 EAMATE data, and will be expanded to include the AWR and perhaps Suspense data, among other files.

The expanded prototype will provide:

- An environment to identify future uses of the technology
- An environment to re-engineer the current scouting processes
- A platform to test uses of the technology and re-engineering concepts
- A mechanism that allows user participation and feedback
- A model of a modernized scouting environment
- A pipeline to migrate prototype concepts to functional specifications for an RFP
- Transference of knowledge to SSA personnel.

---

This system, both hardware and software, is intended solely for use in a prototype environment. The NIST-developed software, while being tested to the best of our abilities, is not guaranteed to be error free. The prototype system design was intended to minimize cost while placing maximum emphasis on establishing the feasibility of an automated system and studying the factors that would impact the design of a full scale system. The prototype system design used does not imply any recommendations for a full scale system design. Any products named are for descriptive purposes and do not imply a recommendation for a final implementation.

---







## **2 Prototype System Components**

The EAMATE prototype system is an integration of NIST-developed software and commercially available software and hardware. The following sections discuss the hardware and software components of the prototype system. Figure 2-1 depicts the system configuration (major components) of the EAMATE prototype system.

### **2.1 Hardware Components**

The EAMATE prototype system file server consists of a SUN SPARCstation 2 with 32 MB of RAM and 3.5 GB of Magnetic Hard Disk. The file server is attached via a SCSI bus to an Hewlett-Packard 88780B 12.7 mm (1/2 in.) open reel tape drive, and an Hewlett-Packard C1710A rack-mountable optical drive library with two 130 mm (5 1/4") magneto-optical drives and 32 cartridge slots.

The workstations used in the prototype are a Compaq 386/25 with 16 MB of RAM and a 300 MB hard drive, and an Epson 386 386SX/20 with an 80 MB Hard drive and 8 MB of RAM.

The file server and the workstations are connected via an ethernet Local Area Network (LAN).

### **2.2 Software Components**

The operating system for the file server during development was SunOS 4.1.2. Later in the prototype development, the operating system was moved to Solaris 2.2 (SunOS 5.2), in order to be compatible with the expanded prototype being installed at SSA. With the exception of the optical disk jukebox, all of the file server peripherals are controlled by the SunOS utilities. The jukebox is controlled by software drivers written by Q-Star Corporation.

The operating system used on the workstations is Microsoft DOS 5.0 and Microsoft Windows version 3.1. Networking on the PCs is handled by the PC-NFS drivers (using version 4.0 for development, and later upgraded to 5.0 for compatibility with the expanded prototype).

The NIST-developed software consists of a user interface which resides on the PC workstations, and indexing and searching applications which reside on the file server. In addition, an entire suite of code was developed to perform the data conversion, and to control and view the data as it was indexed and stored on the optical disk/magnetic media. A set of utilities was developed to operate on the application performance and usage statistics gathered by the user interface code.

The user interface was developed for Microsoft Windows using the C language. The software tools used included: Microsoft C compiler version 6.0, Microsoft Software Development Kit version 3.1, Blue Sky WindowsMAKER Professional version 5.0, Magic Fields version 1.0, and PC-NFS Programmers Tool Kit version 4.0.1.

The index, search and data conversion programs were developed in the C language on the file server. Software tools used to develop this code included: SunOS 4.1.2 C compiler (later ported to the SunPro C compiler version 2.0.1 for Solaris 2.2) and various UNIX utilities.

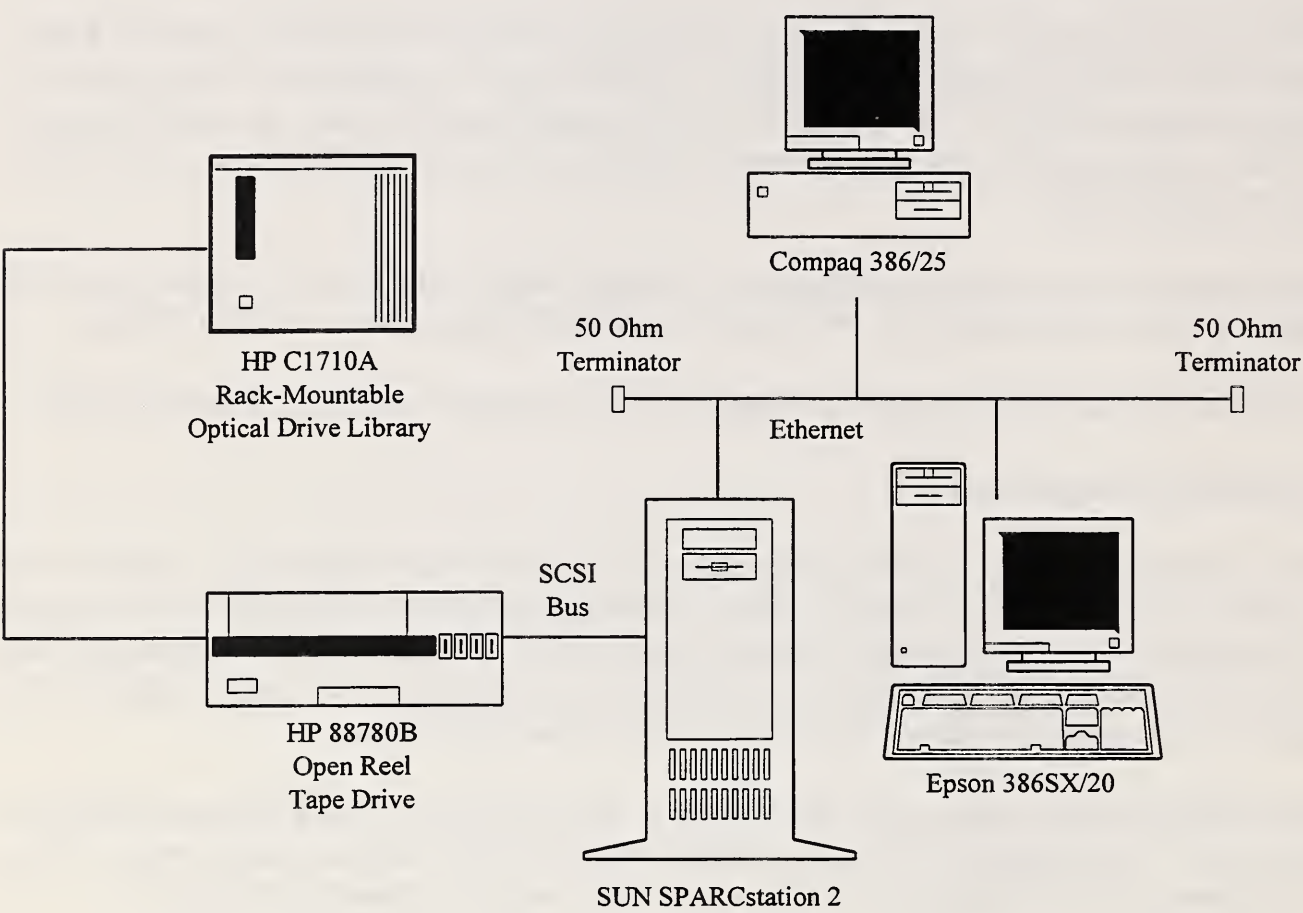


Figure 2-1 EAMATE Prototype System Configuration.

### **3 System and Software Design**

The first section in the following chapter provides a high-level understanding of the basic system design. It describes the role of the user interface and the search engine, and the communication between the two applications. Section 3.2 discusses the aspects of the prototype design which worked particularly well, as well as some areas which were not fully implemented due to the fact that the system is a prototype. Section 3.3 outlines the EAMATE record extraction tool, which, while not implemented in the NIST prototype, will be essential for the full production system. Section 3.4 describes the high level data constructs which are common to all pieces of the software. Further detail on the design of each of the components, and their interaction is given in later sections.

#### **3.1 High Level Design**

At the highest level, the system is designed as a distributed, networked system. The basic components of the system are a file server, which provides storage and retrieval capabilities for the data, and the client workstations, which provide the user with access to the data on the file server. The software installed on the file server includes the data conversion software, the indexing software, and the search software, as well as a suite of software for debugging purposes. The software installed on the client workstation consists of the user interface application and the utilities to operate on the performance statistics.

The file server is the main location for storage of the data. The EAMATE data record is split into two types of data, according to the frequency with which a user accesses the data. The first type of data, the browse data, is determined to be the minimum amount of record data that a searcher would need to identify the target individual. These are the data that are accessed with the highest frequency, and in the largest quantities. The browse data is about 25% of the total data.

The second type of data, the detailed data, is the entire record for an individual. The complete record needs to be accessed much less frequently, often only for one or two individual records out of an entire report. It is necessary to allow access to these data as the information is needed once the desired individual is found, however, it is not necessary to display it for identification purposes.

As the browse data is accessed with high frequency and in large quantities, it is necessary to store it on a fast-access media. Thus, the data are stored on magnetic hard disk. The detailed information is stored on optical media, as it is larger in quantity, but accessed infrequently and in small chunks. More detailed information on the storage strategy can be found in section 7.

The software resident on the file server is responsible for converting the data from Computer-Output to Microfilm (COM) files provided by SSA (the data conversion software), for indexing



the data (the indexing software), and for handling requests for information from the user interface (the search software). The searching software contains many basic routines to provide access to the EAMATE data, as well as routines which implement a unique "best match" search strategy. This search technique allows a searcher to find records which are close to the input query but do not exactly match it, and will rank the matches in relation to their distance from the query. Information on the theory of this search technique can be found in section 5. Section 6 details the design and implementation of the software resident on the file server. The conversion, indexing and searching of the EAMATE data is covered in sections 7 and 8.

The software installed on the client workstation(s), the user interface, is responsible for all communication with the user, and for transmitting requests for information to the file server applications. The user interface acts as the front end application to the file server searching applications. In basic terms, it sends queries to the file server applications and then processes the information that is returned for display and/or hardcopy output. Detailed design information on the user interface is located in section 4.

### **3.2 Prototype Observations**

One of the major goals of the prototype system was to explore high-risk areas of the automation of the EAMATE data. These are determined to be areas that may cause potential problems in a full scale implementation. For example, two of the high-risk areas in the automation of the EAMATE data are the re-engineering of the scouting process to fully take advantage of the capabilities of an automated system, and the development of "best-match" searching techniques to compensate for variations and errors in the data.

As in any prototype system, the design reflects not only concentration in these high-risk areas, but an effort to minimize cost and development time. Due to this, there were several areas that were not implemented as would be expected in a production system design.

This section outlines areas in the design that worked well in the prototype EAMATE system and should be considered for a full implementation. In addition, it outlines areas that were not implemented fully in the prototype system and should not be carried over to a production system.

During the testing of the prototype, one area which had a large impact on the success of the system was the use of the intuitive, windowing environment. Average training time logged during early testing was 1/2 hour. A large majority of the test users had no experience with graphical user interfaces or pointing devices but expressed satisfaction with the system in terms of design, user-friendliness and usability. Use of an intuitive, windowing environment also provided a mechanism for easy access to all relevant data through modular design, understandable dialog and point and click operations.

Another area of high payoff in the automation of the data was the use of the "best-match" search techniques in the indexing and searching applications. The use of this technique allowed the searcher to find the individual he or she was looking for with much greater speed and confidence.

It also turned up matches that would never have been found with traditional search techniques, thus significantly increasing the accuracy of SSA's earnings verification operation.

One other design decision which had a significant impact on the prototype was the division of data into browse data and detail data. This improved the prototype in two ways. The first area was the speed with which the user could browse the data. Not only was the screen not cluttered with useless information, but many more employee matches could be displayed at once, allowing the searcher to process more records per given time period. The second area was in the retrieval speed. Since the browse data is a relatively small portion of the overall data, and was accessed with high frequency, it is effective to store the data on magnetic hard disk instead of on the optical disk in the jukebox. This significantly decreases the retrieval time based upon the throughput rate of magnetic disk vs. optical disk, and the lack of platter swapping that would need to occur in the jukebox.

There were several areas which were not fully implemented in the prototype system. This was due to the fact that they were not considered risk areas for a full implementation, and were therefore not implemented in order to minimize time and money investments. They are as follows:

- In the communication of data between the file server and the client workstations, much use was made of writing files to a shared hard disk. This was done due to communication incompatibilities between several of the development tools that were in use. In a full implementation, passing some of this data implicitly over the network in the procedure calls may eliminate some of the retrieval time.
- In the prototype design, some extraneous data was stored in order to be able to quickly modify the system configuration to test the effectiveness of some of our algorithms. The storage of this extra information would not be necessary in a full system. For example, the detail data stored on the optical disk contains the browse data as well. If the data is stored on magnetic media, there would be no reason to store it as part of the detail information. Also, the data stored in the index and duplicate postings files for the search engine could be substantially reduced by compression and/or bit-packing.
- The communication between the file server applications and the user interface uses Remote Procedure Calls (RPC). This makes the server an iterative server, versus a concurrent server, which could be a bottleneck in a full implementation. For a production system, the use of Berkeley Sockets or System V TLI should be considered to more adequately service concurrent users.
- In the prototype, the use of implicit user numbers was used to distinguish requests from individual workstations. For a production system, something less limiting should most likely be considered.
- Due to a Microsoft Windows limitation, only a certain amount of data can be held in a list box at any one time. This caused a limit of approximately 700 browse records to be viewed at any one time. For a prototype, this was a sufficient number, however, for a full implementation it would be necessary to implement some kind of paging algorithm in order to allow a user to view more data.



- On the prototype system, during a search, the user is presented with an hourglass cursor while waiting for the results of the search to appear. For a production system, a more informative user feedback scheme should be considered, such as telling the user the percentage done, or the relative size of the file that is being searched.
- One of the menu choices in the user interface is "Report Statistics", which is intended to provide overall statistics on the employer report for use in identifying problem reports. This menu choice has not been implemented at this time.
- Some of the file naming conventions in the prototype were adopted because of the need for the client workstations to access the files. As the workstations were using MS DOS as the operating system, the file names were restricted to eight characters and a three character extension, which made some of the file names used rather limited.

### **3.3 EAMATE Record Extraction Tool**

During the development of the prototype system, much interactive work was done with the SSA scouts in order to determine the requirements necessary for the system. While this work was being done, it was identified that there were two very separate areas of scouting existing within SSA, both of which needed access to the EAMATE data. The first area, located in the Division of Certification and Coverage (DCC), accesses the EAMATE data on an individual basis, making corrections to individual earnings. If it is determined that there are problems in a report that are widespread, and affect many employees in the same report, then the report is sent to the Blanket Adjustment Unit (BAU). This unit processes the report as a whole, and makes changes to every employee on the report which meets a certain criteria.

Because not much information was available on the requirements of the BAU scouts, the NIST prototype is geared mainly toward addressing the needs of the DCC scouts. As an initial attempt at providing assistance to the BAU scouts, the "Browse Report" feature is included, which will allow an employer report to be browsed in the order that it was submitted. However, it is recognized that this will not be sufficient for a production system, and, as more requirements have become available from the BAU, the concept of the EAMATE Record Extraction Tool has emerged. A prototype of this tool is currently under development at SSA, and will be installed on the expanded prototype at OCRO when it is completed.

The main focus of the EAMATE Record Extraction Tool is to extract records from an employer report which meet certain user-specified parameters. For example, if the user wished, the tool could pull all employees which have a non-zero FICA tax, and zero FICA wages. This will prevent the user from having to search the entire report for possibly just a few scattered employees. Another requirement of the tool would be to pull employee records randomly from the employer report in order to assess the scope of a problem within the report. The user should be able to specify how many records to view, and should have some way to indicate how they should be scattered throughout the report. As more work is done with the BAU scouts, additional requirements will most likely be determined.



### **3.4 High Level Data Constructs**

The following structure listings are for structures common to the user interface modules, the indexing and searching modules, and the data conversion modules. Other high level structures will be described in the software design section in which they are used.

#### **Employer Header Record**

This structure contains the information in the Employer Header Record. There is one for each employer report, stored at the start of the report. The structure contains the following information:

- Employer Identification Number (EIN)
- Establishment
- Report Year
- Process Year
- Tape Library Number
- Type of Employer
- Name Code
- Other EIN
- Microfilm Reference Number (MRN)
- Ending MRN of Report
- Sequence Number of Report
  
- Employer Name
- Employer Street Address
- Employer City
- Employer State
- Employer Zip Code
- Platter and Side of Report (on Optical Disk)
- Number of Records in Report
- Offset to Final Total Record in Report
- Offset to Cumulative EIN Total in Report

#### **Employer Intermediate Total Record**

This structure contains the information in the Employer Intermediate Total Records. There is one approximately every 11 employees in the employer report. The structure contains the following information:

- Processed Wages
- Reported Wages
- Processed Tips
- Reported Tips

- Processed Wages/Tips/Other
- Reported Wages/Tips/Other
- Processed Federal Tax Withheld
- Reported Federal Tax Withheld
- Processed FICA Tax Withheld
- Reported FICA Tax Withheld
- Processed Earned Income
- Reported Earned Income
- Processed Deferred Compensation
- Reported Deferred Compensation
- Processed Non-Qualifying Wages
- Reported Non-Qualifying Wages
- Control Number
  
- Processed Medicare Wages
- Reported Medicare Wages
- Processed Medicare Tax
- Reported Medicare Tax

### **Employer Final Total Record**

This structure contains the information in the Employer Final Total Record. There is one for each employer report, stored at the end of the report. The structure contains the following information:

- Processed Wages
- Reported Wages
- Processed Tips
- Reported Tips
- Processed Wages/Tips/Other
- Reported Wages/Tips/Other
- Processed Federal Tax Withheld
- Reported Federal Tax Withheld
- Processed FICA Tax Withheld
- Reported FICA Tax Withheld
- Processed Earned Income
- Reported Earned Income
- Processed Items
- Reported Items
  
- Processed Deferred Compensation
- Reported Deferred Compensation
- Processed Non-Qualifying Wages

- Reported Non-Qualifying Wages
- Processed Medicare Wages
- Reported Medicare Wages
- Processed Medicare Tax
- Reported Medicare Tax

### **Employer Cumulative EIN Total Record**

This structure contains the information in the Employer Cumulative EIN Total Record. This is an optional record, and there is at most one for each employer report, stored at the end of the report. The structure contains the following information:

- Processed Wages
- Processed Tips
- Processed Wages/Tips/Other
- Processed Federal Tax Withheld
- Processed FICA Tax Withheld
- Processed Earned Income
- Processed Items

### **Employer Header Information**

This structure is a concatenation of the EAMATE W2 employer header, the EAMATE Final Total and the EAMATE Cumulative EIN total, if it exists. One of these will be created for each report. It is not stored as a part of the report, however, it is stored in a log file with one for each report processed.

### **Employee Detail Record**

This structure contains the information in the Employee Detail Record. There is one for each employee in the employer report, and these records make up the major portion of the employer report. The structure contains the following information:

- MRN for Employee Record
- Social Security Number
- Employee Name
- Pension Indicator
- Deferred Compensation Indicator
- Wages
- Tips
- Wages/Tips/Other
- Federal Tax Withheld

- FICA Tax Withheld
- Advanced Earned Income
- Medicare Wages
- Medicare Tax Withheld
- Control Number
- Street Address
- Dependent Care
- Allocated Tips
- Group Term Insurance
- Uncollected FICA Tax
- City
- State
- Zip Code
- Deferred Compensation
- Statutory Code
- Fringe Benefits
- Non-Qualifying SEQ
- Non-Qualifying NOT

### **Employee Browse Record**

This structure contains a frequently accessed subset of the information in the Employee Detail Record. There is one for each employee in the employer report. The structure contains the following information:

- Social Security Number
- Name
- Wages
- Tips
- FICA Tax Withheld
- Wages/Tips/Other
- MRN of Employee
- Sequence Number of Report
- Wage Type
- Offset to Record Location in Detail Report

### **User Query Structure**

For communication between the user interface and the search engine, a structure of data is used to pass the user's query to the search engine. The structure contains the following information:

- Year
- Employee Identification Number (EIN)



- Establishment
- Sequence Number of Report
- First Name
- Last Name
- Social Security Number
- Offset/MRN

## Record Type Codes

In the Employer Report Detail file, stored on optical disk, each record in the report is prefixed with a one byte character value which identifies the type of record following. This is done because of the variety of the record types in the file, and the need to distinguish between them before reading them. The following is a list of the record types used in the prototype:

EAMATE W2 Employer Header (MATE_W2EH)	0
EAMATE W2 Employee Information (MATE_W2EI)	1
EAMATE W2 Intermediate Total (MATE_W2IT)	2
EAMATE W2 Final Total (MATE_W2FT)	3
EAMATE W2 Cumulative Total (MATE_W2CE)	4
--- Reserved For Later Compatibility -----	
EAMATE W2C Employer Header (MATE_W2CEH)	5
EAMATE W2C Employee Information (MATE_W2CEI)	6
EAMATE W2C Final Total (MATE_W2CFT)	7
AWR W2 Employer Header (AWR_W2EH)	8
AWR W2 Employee Information (AWR_W2EI)	9
AWR W2 Intermediate Total (AWR_W2IT)	10
AWR W2 Final Total (AWR_W2FT)	11
AWR W2 Cumulative EIN (AWR_W2CE)	12
AWR W2C Employer Header (AWR_W2CEH)	13
AWR W2C Employee Information (AWR_W2CEI)	14
AWR W2C Final Total (AWR_W2CFT)	15





## **4 User Interface Design and Implementation**

### **4.1 Reasons for Development**

Following the initial decision to test the feasibility of automating the EAMATE files and to re-engineer the EAMATE portion of the scouting process, the project was divided into two distinct tasks -- the development of new indexing and searching techniques and the development of a highly functional user interface that allowed the users to exploit and interact with these new techniques. Because the user interface serves as the direct link between the user and the data, it was important and necessary to design an easy-to-learn and easy-to-use application that provided quick access to the EAMATE data -- in a format that made sense to the scouts. To achieve these goals, user-centered design with emphasis on usability was employed, and extensive task analysis was conducted.

### **4.2 Overview of the EAMATE User Interface**

To facilitate design of the EAMATE user interface prototype, several information-gathering visits were made to the Metro West branch of the Social Security Administration in downtown Baltimore, MD, including sessions with SSA management and scouts. Information obtained from the visits and meetings, along with a description and observation of the current manual retrieval system, provided the basis for the initial user interface prototype design. Two user testing sessions, each consisting of approximately 25-30 scouts, were conducted and aided in further enhancement and refinement of the user interface prototype. Upon completion of enhancements and refinements, and incorporation of suggestions made by SSA employees, a third testing session was conducted at the Metro West branch of SSA in January 1993. During a six-and-a-half-day period 20 scouts were trained on the EAMATE prototype system. Training times ranged from a half hour to an hour-and-a-half with the average time being one hour. The short learning curve experienced by the scouts illustrates the careful and comprehensive design of the EAMATE user interface prototype. The scouts' comments regarding training and general impressions of the EAMATE prototype are included in Appendix E.

The EAMATE user interface prototype acts as the front end application to the search engine and provides information retrieval and display capabilities along with optional hard copy output for employee record processing, employer report processing, and error report processing. The scout enters search criteria to locate and view a specific employee record and is provided with a display of a set of records based on approximate searching techniques. Detailed information about a specific employee record may then be requested for display or in hard copy format. The user may also enter search criteria to locate and view a particular employer report. The user is again provided with a display of a set of records contained in the specified employer report and has the option to request displayed or printed information about a specific employee record. Additional supporting information is also available from several places within the user interface. During employee record processing, employer report processing and error report processing, employer information and final totals' information may be requested for display or printing. The scout may also request a hard copy of a specific employer report and error report. Key

information, detailed information, and printed information are readily available to the user in an easy-to-understand and easy-to-read format.

The main purpose of the user interface prototype design is to provide a functional, easy-to-learn, visually pleasing format for users requiring quick and accurate access to EAMATE data. Empowerment of the user was the major focus during development of the user interface prototype. As a result of the empowerment focus, maneuverability, consistency, modularity, and accessibility were the major design issues addressed during development - all part of the overall goal of usability. These four criteria provided the building blocks for the re-engineering of the current manual retrieval system regarding the user's interaction with data including organization, format, and display of the data.

### **4.3 Development**

Design and implementation of the user interface prototype was performed on an IBM PC compatible running Microsoft DOS 5.0, Microsoft Windows 3.1, and PC-NFS 4.0 (later upgraded to version 5.0 for compatibility reasons). Microsoft Windows was chosen as an example of an intuitive and user-friendly environment allowing both keyboard and mouse operations. It also ties in nicely with other SSA project plans. PC-NFS is required for the system housing the user interface(client) to communicate with the search engine housed on the server. The user interface prototype sends queries to the search engine and the search engine returns information based on the user request, which is then processed by the user interface.

Construction of the user interface prototype was originally performed using Borland C/C++ and Application Frameworks, WindowsMAKER Professional 5.0, Magic Fields 1.0, and PC-NFS Toolkit 4.01. Borland C/C++ and Applications Frameworks provided the C compiler and specific Microsoft Windows software development tools but has since been replaced by Microsoft C 6.0 and the Microsoft Software Development Kit because of compatibility issues. WindowsMAKER Professional provided visual programming capability and baseline code generation which significantly decreased development time. Magic Fields provided for generation of data validation code which also aided in decreased development time. PC-NFS Toolkit provides necessary functions and libraries required for communication between the user interface and the search engine (PC-NFS was upgraded to version 5.0). Customization of the code relating to search criteria, information retrieval and display, and communications functions was performed by NIST personnel. Structured programming techniques, modular design, and portability issues were addressed and incorporated into the user interface prototype design. Code listings of the user interface prototype are included in Appendix D.

### **4.4 General Design and Screen Characteristics**

All screens of the user interface prototype are displayed in a windowing mode, providing an intuitive and user-friendly environment. Information is available through "point and click" access as well as keyboard-command access. The windows overlap one another but are easily distinguishable through the use of color, size, icons and format. Specific prompts, along with "how-to" information, are incorporated into the different windows further enhancing the user-friendly environment. The user interface prototype may be divided into five basic levels. Level



0 is the main screen which displays five pushbutton choices along with a credit display window. Level 1 contains screens which prompt the user to enter information. Level 2 screens display results of user search/request queries. Level 3 screens display detailed information. The last level of screens, Level 4 screens, generally display information regarding the current process. Error reporting is not limited to specific levels, but is incorporated throughout the interface. Figure 4-1 is a structure diagram displaying the level/screen layout of the user interface prototype.

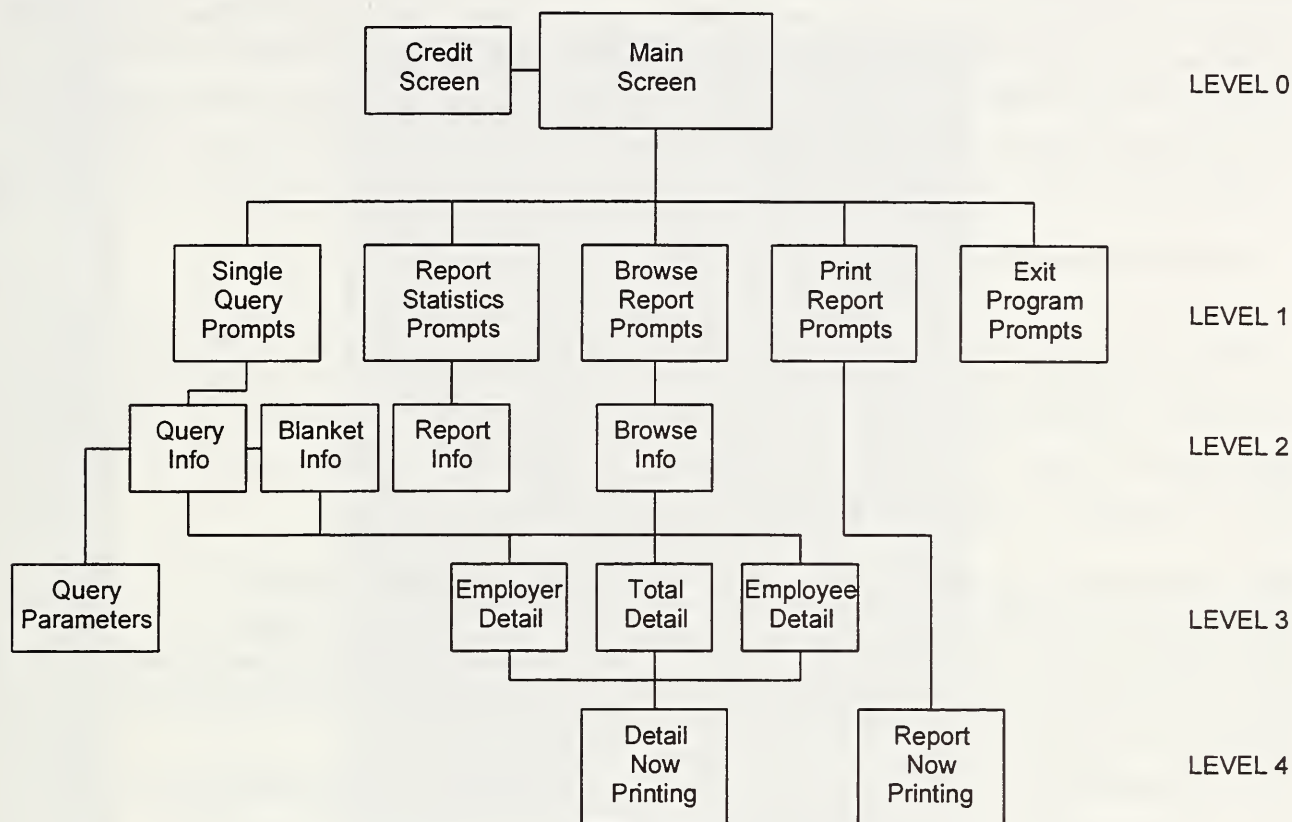


Figure 4-1 Structure Diagram.

To further enhance screen design, colors were chosen to highlight and contrast displayed information and to provide for quick and easy visual discrimination. The color scheme includes blue, green, white, and black with other color accents. The majority of the screens overlay one another as levels of screens are accessed permitting the user to return to his or her previous place without exiting out of specific screens thereby allowing for further selections and operations. The color scheme aids the user by providing a manner for differentiating active and inactive windows and highlighted text. Active windows are surrounded by blue borders and contain black letters on a white background. Inactive windows are surrounded by green borders and also contain black letters on a white background. Highlight bars are blue and contain white text. The color scheme is achieved by setting a color scheme through Microsoft Windows. Other color schemes may be chosen by the user, but will not necessarily provide the contrast and visual discrimination capabilities of the current color scheme.

To provide as much display space as possible, the user interface prototype was developed for a Super VGA monitor. Query screens are organized from a broad to narrow perspective guiding

the query building process as successive levels of the query are defined. The tabular list format of the query fields also provides for a smooth transition between data entry fields. The user may tab to different fields or use the mouse to access a specific field. In some cases the cursor automatically positions itself in the next field after data entry is complete further enhancing ease-of-use. On the information display screens, data is organized in a modular fashion and location consistency is maintained throughout the different information display screens. Employer information is always found at the top of the information screen and employee information is located on the bottom of the screen, organized in a single-line columnar list format augmenting recognition and navigation of the data. The most often viewed fields of employer and employee records are incorporated into the information display screens to facilitate quick access to key information with one or two clicks of the mouse. Pushbuttons that provide access to detail data (or request more data) are positioned along the right side of the information display screens.

#### **4.5 Detailed Screen Descriptions**

This section provides detailed descriptions and illustrations of all the major screens incorporated into the user interface prototype. (As mentioned in the Table of Figures, all SSN's have been removed to ensure data protection.) The interface contains six types of screens: a main screen, query screens, browse screens, detail screens, current process screens, and error reporting screens. Examples of each of the screen types included in this section are:

- |  |   |
|--|---|
| -Main Screen (including the Credit Display Screen) | -Query Information Screen                 |
| -Single Query Prompt Screen                        | -Report Information Screen                |
| -Report Statistics Prompt Screen                   | -Browse Report Information Screen         |
| -Browse Report Prompt Screen                       | -Potential Blanket Information Screen     |
| -Print Report Prompt Screen                        | -Detail Screens                           |
| -Exit Program Prompt Screen                        | -Error, Message & Current Process Screens |
| -Query Parameter Screen                            |   |

##### **Main Screen**

The main screen appears when the application is opened and contains a credit display window and five pushbuttons which allow the user to choose the kind of information query to be performed. The credit display window contains a continue pushbutton, which when pressed, will remove the credit display window and make the five main screen selections available. The five choices are:

- Single Query - used to search for information regarding a single individual
- Report Statistics - used to acquire special aggregate data about a specific employer report (included for the purpose of illustrating functionality only)
- Browse Report - used to pull up a set of records in a specific employer report
- Print Report - used to print a specific employer report
- Exit Program - used to exit the application.

Figure 4-2 depicts the main screen.



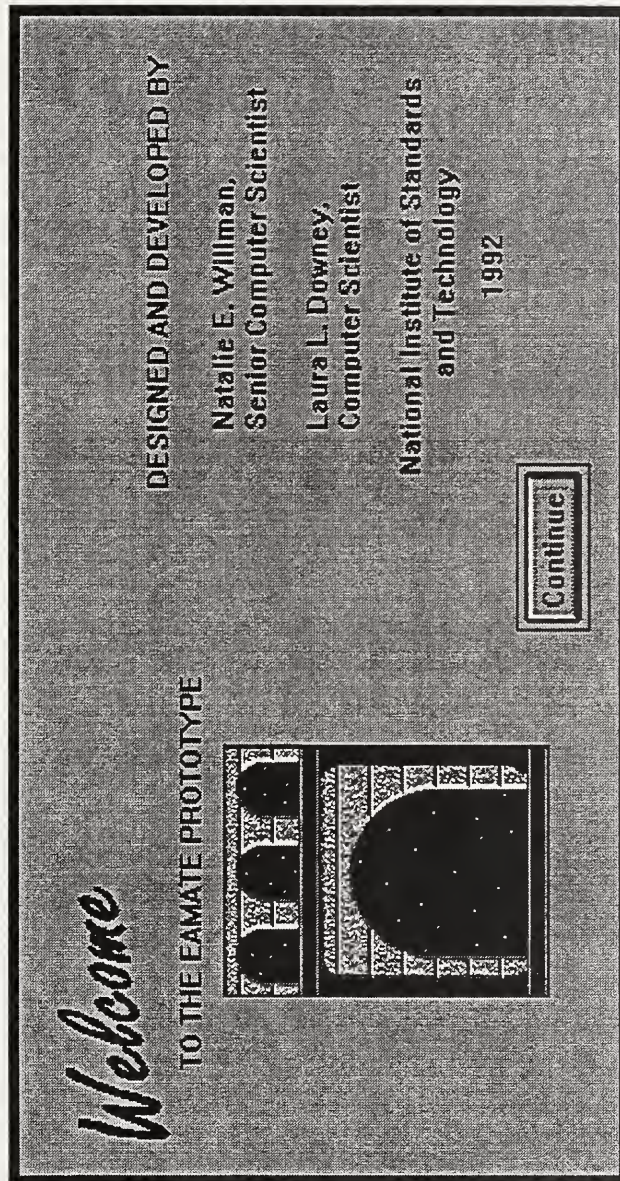


Figure 4-2 Main Screen.



## Single Query Prompt Screen

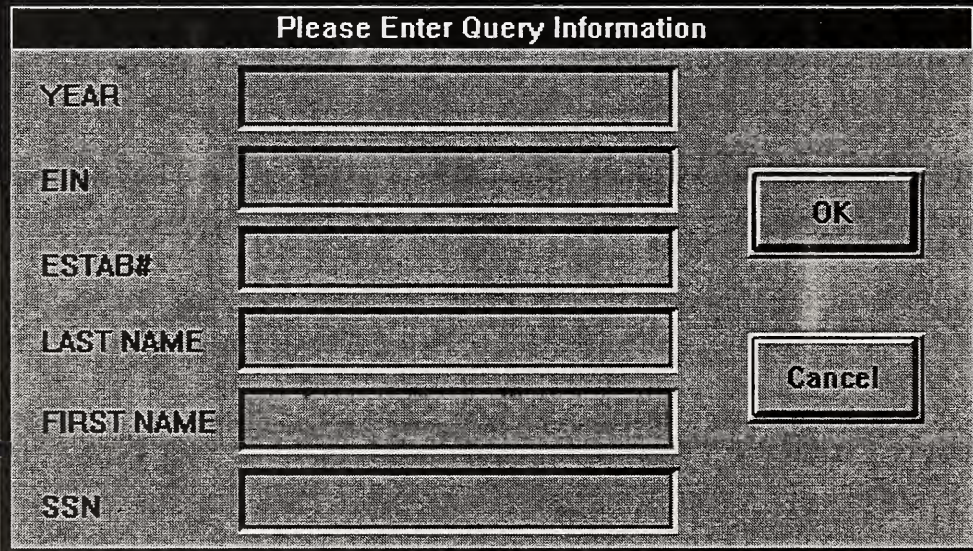
The Single Query Prompt Screen prompts the user to enter information related to a query on a specific individual. The prompts include:

- Year - report year
- EIN
- Establishment Number (optional)
- Last Name of the target individual
- First Name of the target individual
- SSN of the target individual.

Year, EIN and a combination of the First Name, Last Name, or the SSN are the minimum required amount of data to form a single query. Legal combinations of the names and SSN fields are:

- First Name and Last Name
- First Name and Last Name and SSN
- SSN

If the user attempts to move to the next field without first entering data into a required field, an error screen will appear and the cursor will return to the blank required field. If a legal combination of the name and SSN fields is not entered, an error screen will appear. Upon query completion, a request for information may be sent to the search engine via the OK pushbutton or may be canceled via the cancel pushbutton. Figure 4-3 depicts the Single Query Prompt Screen.



The image shows a graphical user interface titled "Please Enter Query Information". It contains six input fields arranged vertically, each with a label to its left: "YEAR", "EIN", "ESTAB#", "LAST NAME", "FIRST NAME", and "SSN". To the right of these fields are two buttons: "OK" and "Cancel". The interface has a dark background with light-colored text and buttons.

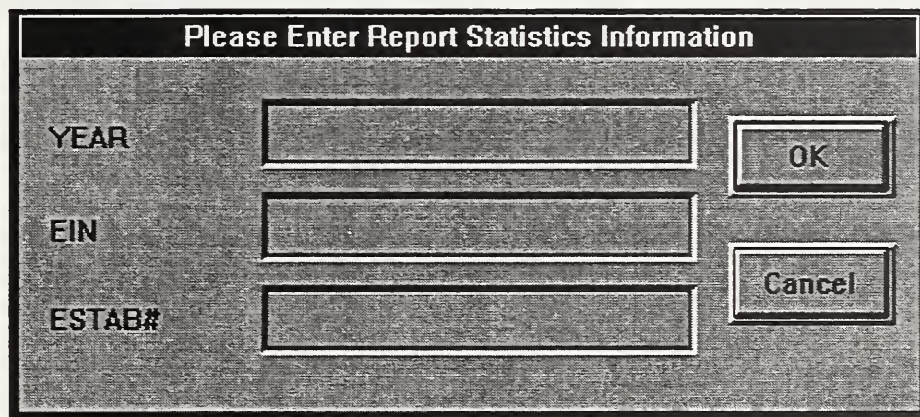
Figure 4-3 Single Query Prompt Screen.

## Report Statistics Prompt Screen

The Report Statistics Prompt Screen prompts the user to enter information regarding a specific employer report. The prompts include:

- Year - report year
- EIN
- Establishment Number (optional)

Year and EIN are both required fields and error screens will appear if the user attempts to move to the next field without supplying information to these fields. When the screen is complete, the report statistics request may be initiated via the OK pushbutton or may be canceled via the cancel pushbutton. (No request is actually sent. This screen is included to demonstrate functionality only and when the OK pushbutton is pressed, a placeholder screen will appear.) Figure 4-4 depicts the Report Statistics Screen.



The image shows a graphical user interface window titled "Please Enter Report Statistics Information". The window has a dark background with a lighter gray area for input. On the left, there are three labels: "YEAR", "EIN", and "ESTAB#". To the right of each label is a rectangular input field. To the right of the input fields are two buttons: "OK" and "Cancel". The "OK" button is positioned to the right of the "YEAR" and "EIN" input fields, and the "Cancel" button is positioned to the right of the "ESTAB#" input field.

Figure 4-4 Report Statistics Screen.



## Browse Report Prompt Screen

The Browse Report Prompt Screen prompts the user to enter information regarding a specific employer report. The prompts include:

- Year - report year
- EIN
- Establishment Number (optional)
- Beginning MRN - the starting point for display of records.

Year, EIN, and Beginning MRN are all required fields and error screens will appear if the user attempts to move to the next field without supplying information to these fields. Upon completion of the browse report query, the request for information may be sent to the search engine via the OK pushbutton or may be canceled via the cancel pushbutton. Figure 4-5 depicts the Browse Report Prompt Screen.

The screenshot shows a dialog box titled "Please Enter Browse Report Information". It contains four input fields arranged vertically, each with a label to its left: "YEAR", "EIN", "ESTAB#", and "BEGINNING MRN". To the right of the input fields are two buttons: "OK" and "Cancel". The "OK" button is positioned to the right of the "EIN" and "ESTAB#" fields, while the "Cancel" button is positioned to the right of the "ESTAB#" and "BEGINNING MRN" fields.

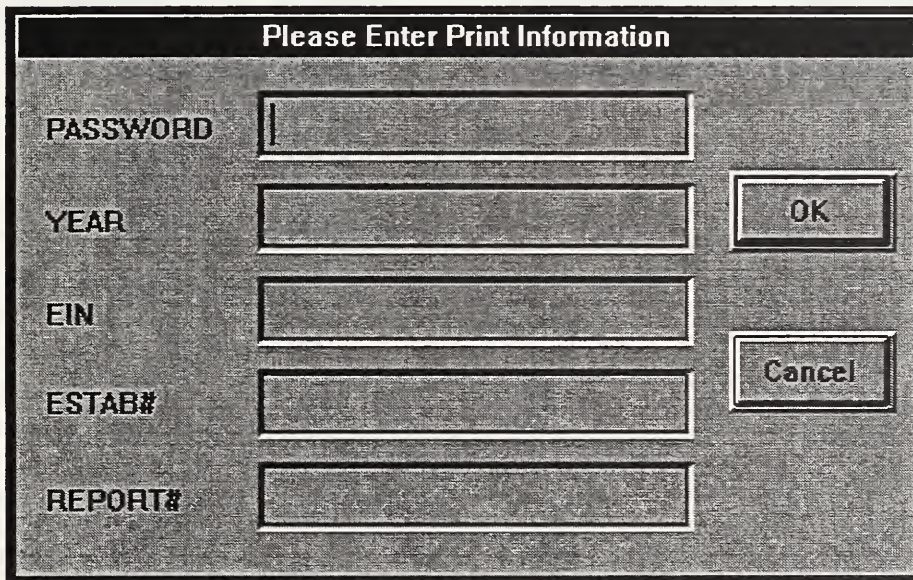
Figure 4-5 Browse Report Prompt Screen.

## Print Report Prompt Screen

The Print Report Prompt Screen prompts the user to enter information regarding a specific employer report for printing. The prompts include:

- Password
- Year - report year
- EIN
- Establishment Number (optional)
- Report Number - the 3-character identifier assigned to a specific report which is part of a multiple-report EIN

The Password (incorporated to prevent unnecessary copies of employer reports [which may be very large] from being printed), Year, EIN, and Report Number are all required fields. Error screens will appear if the user attempts to move to the next field without supplying information to these fields. Upon completion of the print report query, the print request may be sent to the search engine via the OK pushbutton or may be canceled via the cancel pushbutton. Figure 4-6 depicts the Print Report Prompt Screen.



The screenshot shows a dialog box titled "Please Enter Print Information". It contains five text input fields arranged vertically, each preceded by a label: "PASSWORD", "YEAR", "EIN", "ESTAB#", and "REPORT#". To the right of the "YEAR" and "EIN" fields is an "OK" button. To the right of the "ESTAB#" and "REPORT#" fields is a "Cancel" button. The dialog box has a dark border and a light gray background.

Figure 4-6 Print Report Prompt Screen.

## Exit Program Prompt Screen

The Exit Program Prompt Screen prompts the user to make sure he/she wants to exit. Mouse or keyboard operations accomplish the user's intent. Figure 4-7 depicts the Exit Program Prompt Screen.



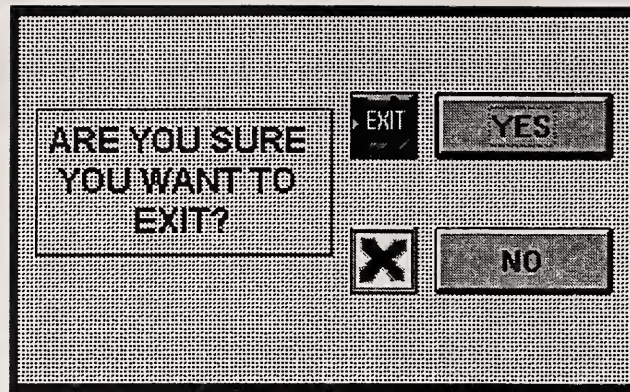


Figure 4-7 Exit Program Prompt Screen.

### Query Information Screen

The Query Information Screen displays a subset of the employer data and a subset of possible matches to the user's query regarding a specific individual. These subsets of data are termed browse data and designed to provide the user with enough data to uniquely identify the employer and/or the employee. (Full descriptions of data are contained in section 4.6.) Matches to the user's request will appear in descending order with the most probable match listed first. If a match is not found for the target individual, additional matches may be requested. A variety of other information is available via pushbutton choices and icons as well as through keyboard commands. Features on the Query Information Screen include:

- Arrow Icon - used to display different employer data when dealing with a multiple-report EIN
- QP Icon - used to display the current query parameters
- Highlight Bar - used to scroll through the possible matches and in conjunction with the enter key to display employee detail data
- Mouse Double-Click - used to display employee detail data
- Employer Detail Pushbutton - used to display employer detail data
- Report Totals Pushbutton - used to display employer report totals
- Potential Blanket Pushbutton - used to access the Potential Blanket Information Screen
- Close Pushbutton - used to close out the Query Information Screen
- Add'l Matches Pushbutton - used to display the next set of 50 possible matches to the user query

As outlined above, the user has the option to request detailed information about the employer, report totals, specific employees, the current query parameters, and blanket information. Help instructions are located in proximity to the desired operations. Closing the Query Information Screen is accomplished via the Close pushbutton or the Potential Blanket pushbutton. Figure 4-8 depicts the Query Information Screen.



# Query Information and Possible Matches

## EMPLOYER BROWSE DATA

RPT-YR: 1991 RPT-NO: AAA EIN: 38-2113393 TYPE: R

MARIAN HEALTH CENTER

P.O. BOX 3168

SIoux CITY , IA 51102



Press  
Arrow to  
Select  
Different  
Report

Employer Detail

Report Totals

Potential Blanket

Close

## EMPLOYEE BROWSE DATA

MRN	Rep. Wage No. Type	FICA Wages	FICA Tips	FICA Tax W/H	Wgs/Tips/ Other	SSN	NAME
10939917085	AAC 0	05004.08	00000.00	0310.24	0005004.08		JANET M SMITH
10939916928	AAA 0	00499.43	00000.00	0030.97	0000499.43		JANET S SMITH
10939917213	AAE 0	24574.68	00000.00	1523.63	0024574.68		JANNET V SMITH
10939920003	AAS 0	24078.40	00000.00	1492.86	0024078.40		D J SMITH
10939919932	AAS 0	20409.47	00000.00	1265.39	0020409.47		B J SMITH
10939919769	AAS 0	42073.60	00000.00	2608.56	0040253.60		P J SMITH
10939919769	AAS 0	08205.96	00000.00	0508.77	0008205.96		M J SMITH
10939919768	AAS 0	03729.88	00000.00	0231.25	0003729.88		K J SMITH
10939919768	AAS 0	33572.48	00000.00	2081.49	0033572.48		K J SMITH
10939919768	AAS 0	03814.95	00000.00	0236.53	0003814.95		J T SMITH
10939919767	AAS 0	01082.32	00000.00	0067.10	0001082.32		B J SMITH
10939919767	AAS 0	15607.16	00000.00	0967.64	0015607.16		B J SMITH
10939919459	AAP 0	19267.49	00000.00	1194.58	0017909.69		J A SMITH
10939919447	AAP 0	23091.10	00000.00	1431.65	0023091.10		D J SMITH

\*\*POSITION THE HIGHLIGHT BAR ON SELECTION AND EITHER DOUBLE-CLICK ON THE BAR OR PRESS THE ENTER KEY TO DISPLAY THE EMPLOYEE DETAIL\*\*



<< Pres QP to Display  
<<< Query Parameters

Add'l Matches

Figure 4-8 Query Information Screen.

## Query Parameter Screen

The Query Parameter Screen displays the most recently entered query parameters. Removal of this screen is accomplished via the OK pushbutton. Figure 4-9 depicts the Query Parameter Screen.



Figure 4-9 Query Parameter Screen.

## Browse Report Information Screen

The Browse Report Information Screen is very similar to the Query Information Screen except that it displays employee browse data in the order that it appears in the report (starting with the MRN provided in the browse query), allowing the user to browse through the report at random. All other features of the Query Information Screen are available on the Browse Report Information Screen except for the Potential Blanket Pushbutton, the Arrow Icon and the QP Icon. Figure 4-10 depicts the Browse Report Information Screen.

## Potential Blanket Information Screen

The Potential Blanket Information Screen is also similar to the Query Information Screen but it displays browse data from 30 records in the employer report -- 10 from the beginning, 10 from the middle, and 10 from the end of the report (matching current procedures for blanket processing). In addition to the attributes of the Query Information Screen (minus the Arrow Icon and the QP Icon) a print option for the Blanket Report is also provided via the Print Blanket pushbutton. Figure 4-11 depicts the Potential Blanket Information Screen.



# Browse Report

## EMPLOYER BROWSE DATA

RPT-YR: 1991 RPT-NO: AAC EIN: 38-2113393 TYPE: R

SISTERS MERCY HLTH CORP.  
C/O MERCY HEALTH CENTER  
DUBUQUE , IA 52001

Employer Detail

Report Totals

Close

## EMPLOYEE BROWSE DATA

MRN	Rep.Wage No. Type	FICA Wages	FICA Tips	FICA Tax W/H	Wgs/Tips/ Other	SSN	NAME
10939917085	AAC 0	12791.19	00000.00	0793.05	0009476.05		CAROL SMART
10939917085	AAC 0	29307.59	00000.00	1817.07	0029307.59		RICKY T SMART
10939917085	AAC 0	01624.67	00000.00	0100.73	0001624.67		ATHENE B SMITH
10939917085	AAC 0	14801.55	00000.00	0917.71	0014151.55		BONNIE SMITH
10939917085	AAC 0	05420.28	00000.00	0336.08	0005420.28		CHRISTOPHER R SM
10939917085	AAC 0	05004.08	00000.00	0310.24	0005004.08		JANET M SMITH
10939917085	AAC 0	29148.80	00000.00	1807.24	0026948.80		JANICE M SMITH
10939917085	AAC 0	15500.13	00000.00	0961.01	0015500.13		PATRICIA L SMITH
10939917085	AAC 0	45542.94	00000.00	2823.66	0036042.96		SALLY J SMITH
10939917085	AAC 0	18342.00	00000.00	1137.23	0018342.00		SUSAN E SMITH
10939917086	AAC 0	05662.50	00000.00	0351.07	0005662.50		JANE A SMOTHERS
10939917086	AAC 0	28751.06	00000.00	1782.55	0026151.06		LYNN A SMOTHERS
10939917086	AAC 0	01610.50	00000.00	0099.85	0001610.50		EDWARD L SNYDEF
10939917086	AAC 0	06705.68	00000.00	0415.74	0006705.68		LINDA K SODERBER

\*\*\*\* POSITION THE HIGHLIGHT BAR ON SELECTION AND EITHER DOUBLE-CLICK ON THE BAR OR PRESS THE ENTER KEY TO DISPLAY THE EMPLOYEE DETAIL \*\*\*\*

Add'l Records

Figure 4-10 Browse Report Information Screen.



# Blanket Information

## EMPLOYER BROWSE DATA

RPT-YR: 1991 RPT-NO: AAA EIN: 38-2113393 TYPE: R

MARIAN HEALTH CENTER

P.O. BOX 3168

SIoux CITY, IA 51102

Employer Detail

Report Totals

Print Blanket

Close

## EMPLOYEE BROWSE DATA

MRN	Rep. Wage No. Type	FICA Wages	FICA Tips	FICA Tax W/H	Wgs/Tips/ Other	SSN	NAME
10939916763	AAA 0	02775.48	00000.00	0172.10	0002775.48		AMY J ADAMS
10939916763	AAA 0	14321.46	00000.00	0887.94	0014321.46		GLENNA M ADAMS
10939916763	AAA 0	22045.72	00000.00	1366.83	0022045.72		PAMELA S ADAMS
10939916763	AAA 0	09207.75	00000.00	0570.89	0009207.75		MICHAEL A ADKISSON
10939916763	AAA 0	21086.51	00000.00	1307.35	0021086.51		ETHEL M AGERSON
10939916763	AAA 0	19161.81	00000.00	1188.05	0019161.81		DEBBIE AHRENDT
10939916763	AAA 0	24821.30	00000.00	1538.89	0024721.30		THELMA B ALBERS
10939916763	AAA 0	15057.20	00000.00	0933.54	0014107.20		LOIS A ALBERTSON
10939916763	AAA 0	00390.00	00000.00	0024.18	0000390.00		SARA J ALBRECHT
10939916763	AAA 0	14498.10	00000.00	0898.91	0014498.10		PETER B ALBRIGHT
10939916861	AAA 0	16359.85	00000.00	1014.34	0016059.85		DALENE KRAY
10939916861	AAA 0	11865.21	00000.00	0735.64	0011145.21		BARBARA J KREBER
10939916861	AAA 0	00179.34	00000.00	0011.12	0000179.34		KEN G KREBER
10939916861	AAA 0	06430.73	00000.00	0398.72	0006430.73		RENEE A KREISEL

\*\*\* POSITION THE HIGHLIGHT BAR ON SELECTION AND EITHER DOUBLE-CLICK ON THE BAR OR PRESS THE ENTER KEY TO DISPLAY THE EMPLOYEE DETAIL \*\*\*

Figure 4-11 Potential Blanket Information Screen.

## Employee Detail Screen, Employer Detail Screen, Totals Detail Screen

The Employee Detail Screen displays complete information as contained in an employee record. The Employer Detail Screen displays complete employer information as contained in the employer header record. The Totals Detail Screen displays complete final totals information for a specified employer report. A hard copy of the detailed information may be requested via the Print Pushbutton and closing any of the detail screens is accomplished via the Continue Pushbutton. Figure 4-12 depicts the Employee Detail Screen, Figure 4-13 depicts the Employer Detail Screen and Figure 4-14 depicts the Totals Detail Screen.

Employee Detail	
SSN:	RPT-NO: AAA
JANET M SMITH 1004 GARFIELD DUBUQUE , IA 52001	
FICA Wages: 05004.08	Pension Indicator: P
FICA Tips: 00000.00	DefComp Indicator:
WgsTpsOther: 0005004.08	Def Comp Amt: 0000000.00
FICA Tax: 0310.24	Sta:
Federal Tax: 0000404.77	Fr Benefits:
Uncollected FICA:	Empr Grp Trm Ins Cost:
Medicare Wages: 005004.08	Dep Care:
Medicare Tax: 0072.56	NQSEC: 0000000.00
Allocated Tips:	NQNOT:
Adv Earn Inc: 00000.00	Ctrl Number: 0001247







Figure 4-12 Employee Detail Screen.







Employer Header Detail		
<b>MARIAN HEALTH CENTER</b> <b>P.O. BOX 3168</b> <b>SIoux CITY , IA 51102</b>		<b>RPT-NO: AAA</b>
<b>EIN: 38-2113393</b> <b>Starting MRN: 10939916763</b> <b>Ending MRN: 10939916960</b> <b>Report Year: 1991</b> <b>Process Year: 1992</b>	<b>Tape Library Number: 234206</b> <b>Other EIN:</b> <b>Estab Number:</b> <b>Type of Employment: R</b> <b>Name Code: F</b>	
   		

Figure 4-13 Employer Detail Screen.

Final Totals		
<b>EIN: 38-2113393</b>		<b>RPT-NO: AAA</b>
TYPE:	PROCESSED:	REPORTED:
FICA Wages	00037125956.13	00037125956.13
FICA Tips	0000000000.00	0000000000.00
Wgs/Tps/Other	00037182000.46	00037182000.46
FICA Tax W/H	0002301776.97	0002301776.97
Medicare Wgs	00037695429.52	00037695429.52
Medicare Tax	00000546499.60	00000546499.60
Number of Items	0002018	0002018
Fed Tax W/H	0004939853.26	0004939853.26
Earned Income	0000000000.00	0000000000.00
DefComp	00000570524.99	00000570524.99
NonEqual	00000000000.00	00000000000.00
 		

Figure 4-14 Totals Detail Screen.

## Error Screens, Message Screens, Current Process Screens

Error and message screens appear when the user has not entered information correctly, or when communication between the user interface and the search engine has failed. Current process screens appear when print requests are being processed. Removal of these screens is accomplished via an OK or Continue pushbutton. Figure 4-15 depicts one example each of an error screen, a message screen, and a current process screen.

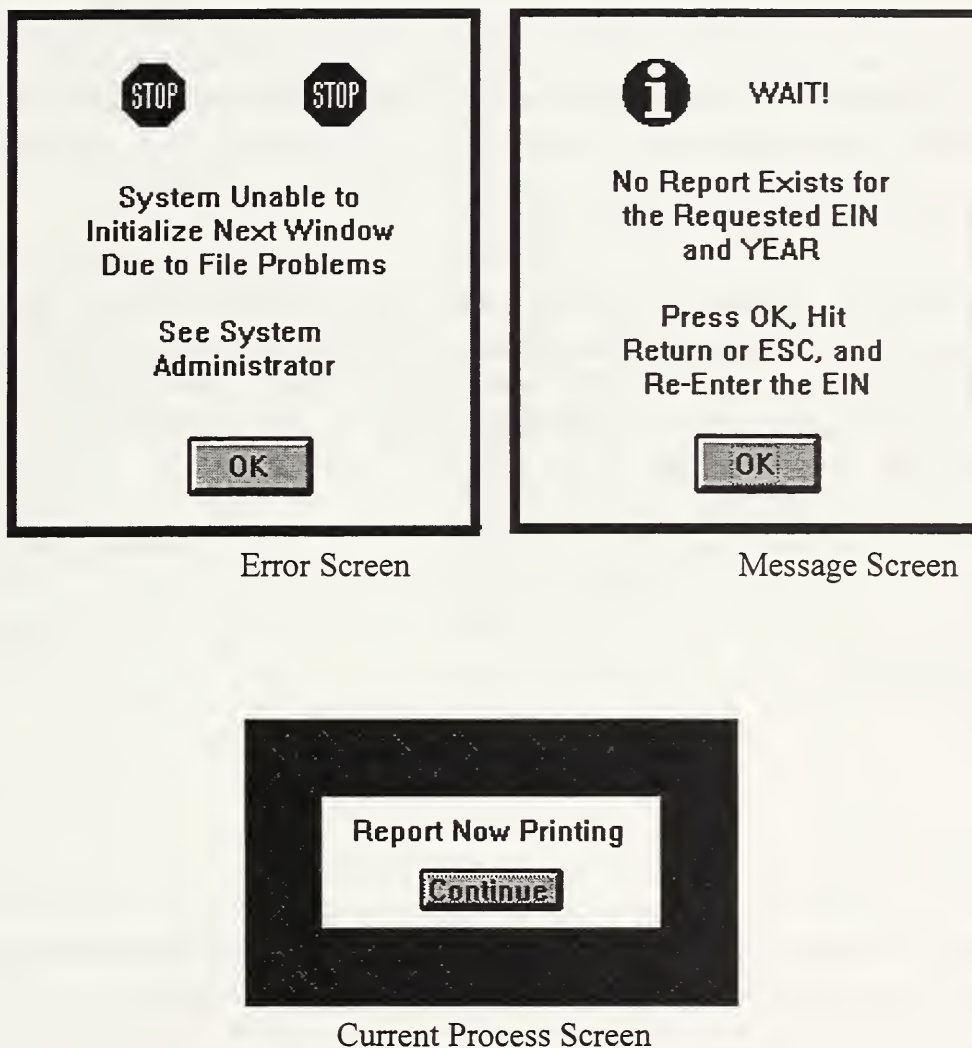


Figure 4-15 Example Error Screen, Message Screen and Current Process Screen.



## 4.6 User Interface Software Organization

This section describes the software data structures utilized by the user interface prototype, both visible (shared with the search engine - see section 3.4) and invisible (only used by the interface). Other components of software organization described here include: function organization, code organization and file usage.

### 4.6.1 Software Data Structures

The user interface prototype uses three major types of data structures: query parameters, browse data, and detail data. Query parameters are used when a request for information is made by the client to the server. Browse data is used to display a subset of information to the user. Detail data is used to provide complete information pertaining to a specific user request. A fourth type of data, statistical data, is also incorporated into the user interface but is basically transparent to the user. Statistical data is discussed in section 10.

#### Query Parameters (visible, except where noted)

Four sets of query parameters exist: single-query, report statistics, browse report, and print report. Each set of query parameters utilizes the user-defined C structure of type query to send and request information to and from the search engine. Each data element of each set is implemented as a set of characters. The baseline structure of the visible type query contains the following definitions:

##### struct query

- Year
- EIN
- Establishment Number
- Report Number
- First Name
- Last Name
- Social Security Number
- Offset/MRN

The four sets of invisible query parameters based on the query structure use only the necessary fields particular to a specific type of query. They contain the following data:

##### squery (Single-Query)

- Year
- EIN
- Establishment Number
- Report Number
- Last Name
- First Name
- Social Security Number

##### rquery(Report Statistics)

- Year
- EIN
- Establishment Number

#### bquery(Browse Report)

- Year
- EIN
- Establishment Number
- Beginning MRN

#### pquery(Print Report)

- Password
- Year
- EIN
- Establishment Number
- Report Number

### **Browse Data (visible)**

The browse data is designed to provide the user with enough data to uniquely identify the employer and/or the employee. It consists of a subset of employer header information and a subset of employee detail information.

The employer header browse information is implemented as a set of characters. The employee browse information is implemented as a user-defined C structure named W2Browse. The employer header browse information and W2Browse contain the following data:

#### Employer Header Browse Information

- Report Year
- Report Number
- Establishment Number
- EIN
- Employer Type
- Employer Name
- Employer Street Address
- Employer City
- Employer State
- Employer Zip

#### W2Browse

- Employee Social Security Number
- Employee Name
- Annual FICA Wages
- Annual FICA Tips
- FICA Tax Withheld
- Annual Wages/Tips/Other
- Microfilm Reference Number
- Report Number
- Wage Type
- Record Location

### **Detail Data (visible)**

Detailed employer information and detailed employee information are stored in user-defined C structures W2EmprInfo and W2EmpInfo, respectively. The employer information is composed of employer header data, final totals data, and cumulative EIN data (if it exists). Separate structures exist for the employer header data, final totals data and cumulative EIN data, but for purposes of communication efficiency, these three sets of data are combined into one structure, W2EmprInfo. Each member of W2EmprInfo and W2EmpInfo is implemented as a set of characters. The two structures contain the following data:

#### W2EmprInfo

- EIN
- Establishment Number
- Report Year
- Process Year
- Tape Library Number

#### W2EmpInfo

- Microfilm Reference Number
- Social Security Number
- Employee Name
- Pension Indicator
- Deferred Compensation Indicator

-Employer Type	-Annual FICA Wages
-Name Code	-Annual FICA Tips
-Other EIN	-Annual Wages/Tips/Other
-Beginning MRN	-Federal Tax Withheld
-Ending MRN	-FICA Tax Withheld
-Report Number	-Advanced Earned Income
-Employer Name	-Medicare Wages
-Employer Street Address	-Medicare Tax
-Employer City	-Control Number
-Employer State	-Employee Street Address
-Employer Zip Code	-Dependent Care
-Platter Side Location	-Allocated Tips
-Initials	-Employer Group Term Insurance Cost
-Number of Records	-Uncollected FICA Tip/Tax
-Browse Start Offset	-Employee City
-Processed FICA Wages	-Employee State
-Reported FICA Wages	-Employee Zip Code
-Processed FICA Tips	-Deferred Compensation Amount
-Reported FICA Tips	-State Employer Code
-Processed Wages/Tips/Other	-Fringe Benefits
-Reported Wages/Tips/Other	-NQSEC
-Processed Federal Tax Withheld	-NQNOT
-Reported Federal Tax Withheld	
-Processed FICA Tax Withheld	
-Reported FICA Tax Withheld	
-Processed Earned Income	
-Reported Earned Income	
-Processed Items	
-Reported Items	
-Processed Deferred Compensation	
-Reported Deferred Compensation	
-Processed Nonequal	
-Reported Nonequal	
-Processed Medicare Wages	
-Reported Medicare Wages	
-Processed Medicare Tax	
-Reported Medicare Tax	
-Flag	
-Cumulative Processed FICA Wages	
-Cumulative Processed FICA tips	
-Cumulative Processed Wages/Tips/Other	
-Cumulative Processed Fed Tax Withheld	
-Cumulative Processed FICA Tax Withheld	
-Cumulative Processed Earned Income	
-Cumulative Processed Items	



## 4.6.2 Function Organization

The user interface is composed of a combination of user-defined C functions, Microsoft Windows API (application program interface) calls and PC-NFS functions. The functions contained in the user interface prototype may be divided into six categories: window creation functions, device control and initialization functions, network functions, string manipulation functions, retrieval and data display functions, and statistical gathering and calculation functions. Each category is listed below. (Standard input/output functions and low-level functions are not listed.) Microsoft Windows API calls are marked with a \* and PC-NFS functions are marked with a \*\*. The user interface prototype uses a variety of remote functions to issue requests to the search engine. These functions, while utilized by the interface, are located on the server and are therefore detailed in section 8.

### Window Creation Functions

BLD_BlankErrDlgFunc()	creates an error message screen
BLD_BlanketErrDlgFunc()	creates an error message screen
BLD_BrowseEntryDlgFunc()	creates Browse Report Prompt Screen
BLD_BrowseReportDlgFunc()	creates Browse Report Information Screen
BLD_DataErrDlgFunc()	creates an error message screen
BLD_DFileErrDlgFunc()	creates an error message screen
BLD_EINErrDlgFunc()	creates an error message screen
BLD_EINorSeqErrDlgFunc()	creates an error message screen
BLD_EmployeeDetailDlgFunc()	creates Employee Detail Screen
BLD_ErrorFileDlgFunc()	creates an error message screen
BLD_FunctionDlgFunc()	creates Credit Display Screen
BLD_Function2DlgFunc()	creates the Exit Screen
BLD_Function5DlgFunc()	creates Report Now Printing Screen
BLD_Function6DlgFunc()	creates Blanket Information Screen
BLD_GetNumCopyDlgFunc()	creates How Many to Print Screen (employee detail)
BLD_HeaderDetailDlgFunc()	creates Employer Detail Screen
BLD_HFileDlgFunc()	creates an error message screen
BLD_MAINCIfunc()	creates the Main Screen
BLD_MatchErrDlgFunc()	creates an error message screen
BLD_MissingFileDlgFunc()	creates an error message screen
BLD_MRNErrDlgFunc()	creates an error message screen
BLD_NMountDlgFunc()	creates an error message screen
BLD_NMSGDlgFunc()	creates an error message screen
BLD_NoMoreMatchesDlgFunc()	creates an error message screen
BLD_NULLPtrDlgFunc()	creates an error message screen
BLD_OKDlgFunc()	creates the Query Info. & Possible Matches Screen
BLD_PrintBlanketDlgFunc()	creates Blanket Report Now Printing Screen
BLD_PrintDlgFunc()	creates Print Report Prompt Screen
BLD_PrintEmpDetailDlgFunc()	creates Employee Detail Now Printing Screen
BLD_PrintHeaderDetailDlgFunc()	creates Header Detail Now Printing Screen
BLD_PWErrDlgFunc()	creates an error message screen

BLD_QparamDlgFunc()	creates Query Parameters Screen
BLD_QUERYDlgFunc()	creates Query Prompt Screen
BLD_QueryErrDlgFunc()	creates an error message screen
BLD_QueryMessageDlgFunc()	creates an error message screen
BLD_QueryTxtDlgFunc()	creates an error message screen
BLD_QuestDlgFunc()	creates User Questions Screen
BLD_ReportStatisticsDlgFunc()	creates Report Statistics Screen
BLD_ReportDlgFunc()	creates Report Statistics Prompt Screen
BLD_ReportTotalsDlgFunc()	creates Report Totals Detail Screen
BLD_SeqErrDlgFunc()	creates an error message screen
BLD_SequenceDlgFunc()	creates Report Number Prompt Screen
BLD_StatEmptyDlgFunc()	creates an error message screen
BLD_StatOpenDlgFunc()	creates an error message screen
BLD_StatWOpenDlgFunc()	creates an error message screen
BLD_StatWriteDlgFunc()	creates an error message screen
BLD_SysErrDlgFunc()	creates an error message screen
BLD_TotNowPrintDlgFunc()	creates an error message screen
BLD_UserNumErrDlgFunc()	creates an error message screen
BLD_Year_ErrDlgFunc()	creates an error message screen

## Device Control and Initialization Functions

AbortProc()	performs print management
BLD_ApplicationAppInit()	initializes application for use with Magic Fields
BLD_QuitFuncUDCFunc()	quits the application
GetPrinterDC()	creates the device context for the default printer
SendDlgItemMessage()*	sets or gets display parameters for a child window

## Network Functions

clnt_call(**)	calls a remote function across the network
clnt_control(**)	sets certain network parameters remotely
clnt_create(**)	creates a client handle to a remote function
clnt_destroy(**)	destroys a client handle

## String Manipulation Functions

CopyBlanket_EDetail()	copies blanket information to employee detail
CreateEDetailString()	creates employee detail information
CreateHDetailString()	creates employer detail information
CreateHeaderString()	creates employer browse information
CreatePrintEDetail()	creates employee information for the printer
CreatePrintTotString()	creates final total information for the printer
CreateTitlePage()	creates title page for the blanket report
CreateTotalString()	creates final totals detail information



## Retrieval and Data Display Functions

GetDlgItem()*	gets a handle to a child window
GetDlgItemText()*	gets input from a child window
SendDlgItemMessage()*	displays string information in a child window
SetDlgItemText()*	displays string information in a child window

## Statistical Gathering and Calculation Functions

Fill_Aggregate()	fills and calculates aggregate statistics
Fill_Current()	calculates times for current statistics
Init_Aggregate()	initializes aggregate statistics structure
Init_Current()	initializes current statistics structure
Write_Aggregate()	writes aggregate statistics to a file

### 4.6.3 Code Organization

The user interface prototype code was developed using various tools including code generators, visual programming resources, data validation tools, communication toolkits, and standard function libraries in addition to the custom code designed and developed by NIST personnel. This section describes the major code modules and necessary supporting code required to put the application together. Standard libraries and standard header files are not discussed but descriptions of these may be found in the appropriate commercial documentation. Code listings are located in Appendix D.

The code may be divided into four basic groups: resource files, source files, header files and production files. Resource files contain definitions for resources used by functions in the source files and are identified by an .rc and .dlg file extension. The source files contain the logic of the program and are identified by a .c or a .wmc file extension. The header files contain definitions and prototypes and are identified by the standard .h file extension. The production files designate how the compiler and the linker should build the executable and are identified by a .def or .lnk file extension. Specific code modules and supporting files are described below.

#### Resource Files

EAMAT42.RC contains resource definitions for all the resources used by the user interface prototype including bitmaps, icons, cursors, and fonts.

DIALOG.DLG contains dialog box definitions for all of the dialog boxes used by the user interface prototype.



## **Source Files**

CUSTOM.C contains string data manipulation functions.

EAMAT42.C contains the main window function, the credit display code and instructions for setting up the message loop for the application.

ERROR.C contains custom error message dialog box functions.

PRINT.C contains code for printing data and for displaying print messages to the users.

SERVICE.C contains core functions for displaying standard error messages to users, scrolling dialog boxes, controls and toolbars in a window, processing help key requests, and displaying bitmaps and icons as decorations, backgrounds, and automated buttons.

STAT.C contains functions that initialize, calculate and record measurements and statistics.

USERCODE.C contains dialog box functions that handle user queries and display information to the user. These include all data entry screens and data display screens. This file also contains instructions for initializing and exiting the application. The major portion of custom code written by NIST personnel was added to this file to achieve the desired functionality of the application.

All .WMC files contain default dialog box function processing for corresponding code modules and are maintained by the code generator.

## **Header Files**

GENERIC.H is the main header file for the application and is maintained by the code generator. It contains constant ID's, definitions, function prototypes and programmer-designated header files based on programming specifications during development of an application.

GLOBAL.H contains custom global definitions for usercode.c. Specifically, this header file contains user-input variables, string variables and structure variables that are used across different functions in usercode.c

GLOBAL1.H contains external declarations which make global definitions for usercode.c and other modules available to all code modules including user-input variables, string variables, credit-window variables, structure variables and custom function prototypes.

MAIN.H defines variables to be used to insert credits in the client area of the main window.

STRUCT.H contains record type definitions for the EAMATE data along with query and browse data type definitions. It also contains statistical structure type definitions.

## **Production Files**

EAMAT42.DEF and EAMAT42.LNK are production files that contain compiler and linker instructions for the building of the executable.

### **4.6.4 File Usage**

The user interface prototype software uses several files during execution. These files are used to record communication and system errors, application performance and usage statistics, and user query data and search results data. Below is the name of each file, the file type, and a brief description of what the file contains and/or how it is used. (NOTE: The # sign represents the user number where it appears in the file names listed below.) Three files, err.fil, usernum.fil and stat#.fil are local to the application directory. All other files are located on the server in the directory export/home/ssa which is mounted to the client as the d: drive (through PC-NFS). The server and the client "share" this drive and the associated files, in essence using the drive/files as the main communication link. Other references to file usage by the prototype may be found in the sections 3, 9, and 10 and in Appendix C.

#### **Local Files**

ERR.FIL is an ascii file which contains error messages describing communications and system errors. It is included as a system administration tool and a troubleshooting aid.

STAT#.FIL is a binary file used to record application performance and usage statistics structures. It is operated on by a DOS executable to interpret and record the statistics in an ASCII format for analysis purposes.

USERNUM.FIL is an ascii file containing the user number that uniquely identifies each client. Each time the application is opened, it accesses usernum.fil and assigns the user number to that specific client.

#### **Shared Files**

BLANK#.TXT is an binary file containing 30 employee records (10 from the beginning, 10 from the middle, and 10 from the end) from the specified employer report. The search engine records the data in the file after a potential blanket request is logged and the user interface interprets the data and displays it.

BROW#.TXT is an binary file that contains browse data matching the most recent user request for single query or browse report information. Once the file is written by the search engine software, the user interface software interprets the data and displays it.

DETL#.TXT is an binary file containing detailed employee information. After receiving the appropriate request, the search engine software records the data and the user interface software interprets the data and displays it.

HDR#.TXT is an binary file containing user-requested information about an employer including detailed data, totals data and cumulative EIN information if it exists. Following a request for employer data, the search engine software writes the file and the user interface software interprets and displays the data.

QUERY#.TXT is an binary file used to record user query parameters. It is written by the user interface and utilized by the search engine to interpret data requests.

#### 4.7 Phase Transition in the User Interface

The user interface software behavior may be partitioned into five distinct phases: the initialization and control of the user interface environment, the entry of the user query, the selection of matches or selection of a report in answer to a query, the display and viewing of browse information and the display and viewing of detail information . The different phases are discussed in the following sections. Figure 4-16 graphically depicts the phase transition of the user interface software.

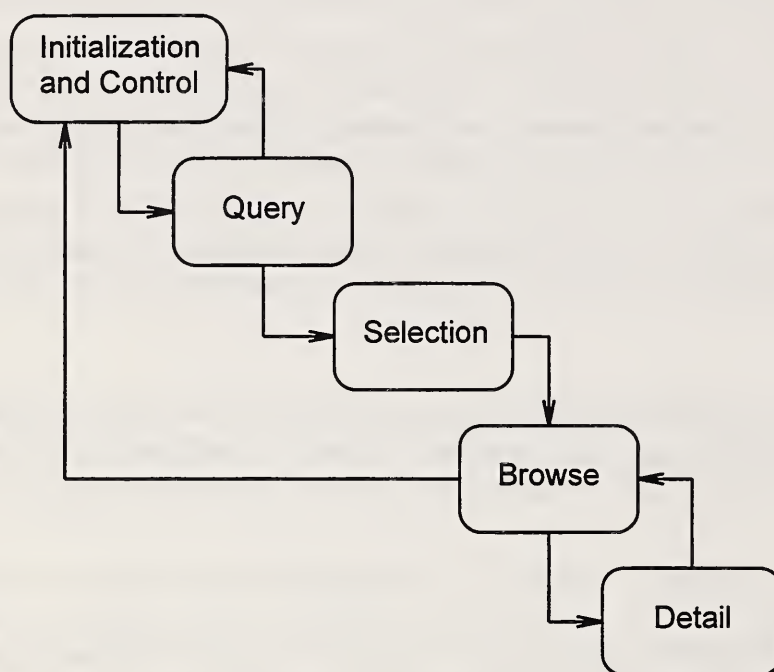


Figure 4-16 Phase Transition Diagram.

##### 4.7.1 Initialization and Control of the User Interface Environment

The user interface is designed as a Microsoft Windows' application, therefore windowing, screen handling, cursor control functions, and mouse and keyboard operations are fully incorporated into the user interface prototype code and handled per Microsoft Windows standard procedures. The WinMain() function of the user interface prototype initializes and opens the application displaying the main screen at full size. WinMain() also sets up a messaging procedure that will



receive messages from the other windows that are used during the current session and will also ultimately send a quit message to the Microsoft Windows messaging loop upon exit of the application. After WinMain() performs initial set up, BLD\_MAINCFunc() is called which creates and displays the main screen. Additionally, BLD\_FunctionDlgFunc() is called which displays the credit window. Upon closing of the credit window, BLD\_FunctionDlgFunc() reads the user number and stores it for future use. The main screen is now ready to accept input.

From the main screen, the user may choose from the five options displayed. The first four options allow the user to enter the query phase, while the last option permits the user to exit the application.

When the exit option is chosen, BLD\_Function2DlgFunc() is called which creates the Exit window. The user may choose to either exit the application or return to the main screen. If the user chooses to exit the application, the function BLD\_QuitFuncUDCFunc is called and the application exits with a zero to the Microsoft Windows message control loop. If the user chooses to remain in the application, BLD\_MAINCFunc() is called, and control is returned to the initialization and control phase.

#### **4.7.2 The Query Phase**

Four distinct options allow entry into the query phase. If the user chooses the single query option, BLD\_QUERYDLgFunc() is called, which creates the Single Query Prompt Screen and captures the user input. Upon successful completion of the Single Query Prompt Screen the client sends a request for employer header information and for matches to the target individual.

If the user chooses the report statistics option, BLD\_ReportDlgFunc() is called, which creates the Report Statistics Prompt Screen and captures the user input. Upon successful completion of the Report Statistics Prompt Screen, a place-holder screen is displayed. The Report Statistics Display Screen (the place-holder) is included in the application for the sole purpose of demonstrating functionality. No implementation exists at this time.

If the user chooses the browse report option, BLD\_BrowseEntryDlgFunc() is called, which creates the Browse Report Prompt Screen and captures the user input. Upon successful completion of the Browse Report Prompt Screen the client sends a request for employer information and for report information.

If the user chooses the print report option, the function BLD\_PrintDlgFunc() is called, which creates the Print Report Prompt Screen and captures the user input. Upon successful completion of the Print Report Prompt Screen the client sends a print request.

Before full completion of any of the query screens, the user has the option to cancel the operation and return to the main screen or to send requests for information and proceed to the selection phase (except for Report Statistics).

### 4.7.3 The Selection Phase

Upon completion of the query phase, the application enters the selection phase which consists of selection of matches to a user query, and selection of a specific report (or information about a report) requested by the user. The selection phase is accomplished by remote procedure calls (RPC) to remote functions located on the server. These calls are:

- browse\_report() - used to obtain sections of the employer report based on the MRN
- get\_blanket() - used to obtain 30 employee records for blanket processing
- get\_seq\_header() - used to obtain specific employer and totals information
- print\_report() - used to send a request to print a specific report
- single\_query() - used to obtain a set of matches to a user query

The selection process is performed by the search engine and is discussed in detail in section 8 of this report. Upon completion of the selection phase, the user enters the browse information phase.

### 4.7.4 The Browse Information Phase

Depending on the choice made during the query phase, different functions are called during the browse information phase.

If the user has just completed a single query, BLD\_OKDlgFunc() is called, which displays the Query Information and Possible Matches Screen. This screen displays possible matches to the user query, browse information and, also offers access to various types of data. The processing and subsequent display of data is accomplished by reading files that contain matches to the user query and other associated related data, customizing the data format, and using MS Windows API calls to display information. One feature included in the Query Information and Possible Matches Screen is a Potential Blanket Option. If the user selects this option, BLD\_Function6DlgFunc() is called, which creates the Blanket Information Screen and displays a set of 30 employee records with associated browse information, and, also offers access to the same type of information located on the Single Query and Possible Matches Screen.

If the user has just completed the report statistics query, the function BLD\_ReportStatisticsDlgFunc() is called, which displays the Report Statistics Screen (as previously referenced, a place-holder screen).

If the user has just completed a browse report query, the function BLD\_BrowseReportDlgFunc() is called, which displays the Browse Report Information Screen, including a list of employees contained in that report; and, also offers access to various types of data. This is accomplished by reading a file containing specified data, customizing the data format, and using MS Windows API calls to display information.

If the user has just completed a print report query, the function BLD\_Function5DlgFunc() is called, which displays a message indicating the report is now printing. During the browse



information phase, a certain amount of set up is done in preparation for the detail phase. To make employee detail information available in the detail phase, an index and record offset of the browse information is obtained and used to compose a remote procedure detail request to the server.

Upon completion of the browse information phase, the user may return to the main screen and enter the query phase or exit, or the user has the option to proceed to the detail phase.

#### **4.7.5 The Detail Phase**

The detail phase may only be entered from within the browse information phase. Three types of detailed information exist: employer, totals, and employee. Employer and totals information are readily accessible from the structure CurrEmprInfo, which has previously been filled by a request for employer information. To display detailed employer information, the function BLD\_HeaderDetailDlgFunc() is called, which creates the Employer Detail Screen. To display detailed totals information, the function BLD\_ReportTotalsDlgFunc() is called, which creates the Report Totals Screen. To display detailed employee information, a remote procedure request is made using an index and record offset obtained from the browse information. The function BLD\_EmployeeDetailDlgFunc() is then called, which creates the Employee Detail Screen.

Upon completion of the detail phase, the user is returned to the browse information phase.

#### **4.8 Re-Engineering The Process**

The user interface prototype design contributes to the re-engineering of the current manual process by providing easy access to the data and by providing a more functional data format. The list box that contains the browse information is much easier on the eyes than scrolling through microfilm. The user can easily identify the current selection through use of the highlight bar, and may easily scroll through the data with a click of the mouse. By scrolling through the browse information instead of full detail data, the scout is able to process more employee records per given time period. The columnar format of the browse data allows the scouts to identify mistakes that otherwise would not have been noticed using the current process. Separation of the data into browse data and detail data allows quicker identification of a target individual while searching through a minimum amount of data. However, the user may easily access the full employee or employer record and view the detail data when necessary. Detail data is also displayed in a modular format to facilitate determination and discrimination of specific fields. All pertinent information regarding a query is literally at the user's fingertips. Identical access and display methods are incorporated into the different levels of the user interface to provide consistency and usability. Information is accessed in the same manner within the different screens making it unnecessary for scouts to learn different access methods when navigating the application.

Besides the obvious benefits of automating the EAMATE portion of the scouting process, the interactions described here demonstrate the re-engineered process in terms of reorganizing the data and changing the ways in which the users interact with the data, thus permitting individuals



to perform their jobs in a more productive and efficient manner. User-centered design with emphasis on usability, task analysis, and the key design criteria of maneuverability, consistency, modularity and accessibility all contributed to the re-engineering effort.

#### **4.9 Quality Assurance Measures**

A variety of measures were taken to support quality assurance. The user interface was designed using a modular format, structured design techniques (including top-down design and step-wise refinement), and extensive internal documentation to facilitate the flexibility, testability and maintainability of the software. Code testing was performed at each stage of development including parameter checking, function return-value checking, data validation, functionality correctness, and pointer and handle validation. To ensure the prototype is utilized to its fullest and in the appropriate manner, NIST personnel incorporated a variety of error trapping and reporting techniques from message beeps and message boxes to recording communications and data errors in a file when system errors occur.

Employing a prototyping method along with an iterative design process, including consultation with users at various phases and user testing, significantly aided quality assurance efforts. User-centered design and thorough task analysis contributed to the ease of use and the understandability of the software. Both are direct results of the application of quality assurance techniques such as consistency, modularity, accessibility, usability and correctness. Inclusion of software quality assurance techniques in the design and development process has helped create a highly-functional, easy-to-use and easy-to-learn prototype application that makes sense to the users of the software.

## **5 Design of the "Best-Match" Search Algorithms**

The following sections describe the reasons why the search engine was developed, the previous research that was built upon in the development of the search engine algorithms, and gives a detailed technical description of the algorithms used in the search engine. In addition, sections are included detailing the laboratory results gathered in the performance evaluation of the searching methodology, and areas for future research which may yield improvement.

### **5.1 Reasons for Development**

Traditional database searches are performed by finding an exact match between an input query and one or more fields of the database. Flexibility can be introduced into the search by allowing Boolean operators to search over several fields, and wild card characters to match subsets of fields. This method works well for databases with few errors, or when the data in the fields are uniform. However, there are some databases where the field to be searched upon contains many errors, or the format of the field is uncontrolled, and an exact match would be impossible. In these instances, it is necessary to be able to retrieve records which are a close match to the input query, similar to methods used in traditional information retrieval of full text data.

The particular set of data for which this methodology was developed has many errors in both the social security number and name fields, and the name field has no fixed structure. Therefore, in addition to misspellings, the name field consists of random permutations of a first name or initial, a middle name or initial (sometimes omitted), a last name, various punctuation marks, and white space. In addition, there are often titles or qualifiers in the field (i.e., Mr., Mrs., Dr., Jr., III, deceased, etc.). Apart from these discrepancies, the nature of the use of the data introduces other complications in the search. This includes such things as reconciling maiden names and married names, and identifying records based upon half of a hyphenated name.

The goal of the retrieval system was, therefore, given a search query which included a personal name, to be able to quickly retrieve all records with name keys within a certain proximity of the search query name, and to rank the records in terms of their similarity to the query name. The records returned by the search are then browsed for the proper match, and additional information for the chosen record is retrieved from an optical disk.

### **5.2 Previous Research in Best-Match Searching**

In a survey of personal name matching applications by Borgman and Siegfried (1992), all the systems examined used algorithms based on phonetic matching and/or pattern matching to judge the similarity between names. These two techniques are also widely used in string matching algorithms. In a phonetic algorithm, strings are judged alike by their similarity of sound. Pattern matching algorithms, on the other hand, detect similarity based upon the distribution and combination of character patterns in strings.



Rogers and Willett (1991) tested several string matching methods using spelling correction for the retrieval of historical word forms from a modern word form. These included reverse error algorithms (Damerau, 1964) and phonetic coding algorithms such as the Soundex (Russell, 1918; Russell, 1922), Phonix (Davidson, 1962; Gadd, 1988), and Davidson (Davidson, 1962) codes, and a tailored Phonix code called the Hartlib Code (Rogers & Willett, 1991). In these experiments, they achieved recall rates of at least 65% and as high as 94%. In a later experiment by Robertson and Willett (1992), three more string matching techniques were explored: n-gram matching (Angell, Freund & Willett, 1983), non-phonetic coding using SPEEDCOP (Pollock, 1980; Pollock, 1981), and dynamic programming using the Wagner-Fisher algorithm (Wagner & Fisher, 1974). Recall rates for these series of experiments ranged from 76% for SPEEDCOP to 96% for the Wagner-Fisher algorithm, with digrams reaching 95%. However, the Wagner-Fisher Algorithm required about 30 times as much processing time as the digram matching (Robertson & Willett, 1992).

Based upon the high recall rates exhibited in the above set of experiments, n-gram pattern matching was chosen for use with this collection of data. In order to optimize the recall, several experiments with n-gram size and overlap were run. In addition, since the data files were large, and an interactive system was required, it was necessary to incorporate an indexing strategy. This strategy indexed the n-gram combinations using an inverted file combined with an instance file, a methodology which was loosely based upon the Harman and Candela document retrieval system (Harman & Candela, 1990).

### 5.3 Technical Description of Algorithms

The following sections describe the basics of n-grams, and describe in detail the indexing, searching, and pruning methodologies.

#### 5.3.1 N-grams

An n-gram is a subset of characters of length n taken from a string of at least length n. For this system, the n-grams derived from a string were adjacent characters, and contained no punctuation or white space. Two lengths of n-grams were tested: digrams ( $n = 2$ ) and trigrams ( $n = 3$ ). In addition, overlapping n-grams and non-overlapping n-grams were tested. For example, the name "BARNES" would be split as follows:

digrams, overlapping	"BA", "AR", "RN", "NE", "ES"
digrams, non overlapping	"BA", "RN", "ES"
trigrams, overlapping	"BAR", "ARN", "RNE", "NES"
trigrams, non overlapping	"BAR", "NES"

Using n-grams, the similarity of two names is assumed to be a function of the number of matching n-grams between the words, i.e. the more n-grams a record has in common with the search name, the higher its rank would be in the search results. The similarity function used for these experiments is the Dice Coefficient (Perry & Willett, 1983).



### 5.3.2 Indexing Methodology

A common file structure used for indexing data files is the inverted file. In order to minimize costly disk accesses when searching external files, the inverted file is often implemented using a B-tree (Knuth, 1973). This algorithm utilizes a B+ tree (Cormen, Leiserson & Rivest, 1990), where all keys are stored in the leaves of the B-tree, together with an instance file for duplicate key matches to form the backbone of the index. The use of the instance file dramatically decreases the size of the index, as B-trees require a rather high storage overhead. The B+ tree contains a key for each n-gram permutation in the file, and, for each key, an instance count -- the number of records containing this n-gram -- is maintained, as well as a linked list of data record offsets and the corresponding record weights (the number of n-grams in the record). This structure is derived from the structure of the Harman and Candela document retrieval system (Harman & Candela, 1990).

During the indexing process, as each record is read from the data file, it is split into n-grams. Each of these n-grams is searched in the B+ tree, and, if found, the instance count for the n-gram is incremented and the data offset and record weight is added to the linked list for the n-gram. If the n-gram is not found, it is added to the tree, the instance count is initialized, and the linked list of record offsets is created. To speed the indexing process, a new node is always added to the beginning of the linked list to prevent having to walk the entire list for each addition. Figure 5-1 shows the structure of the B+ tree index files and the instance files.

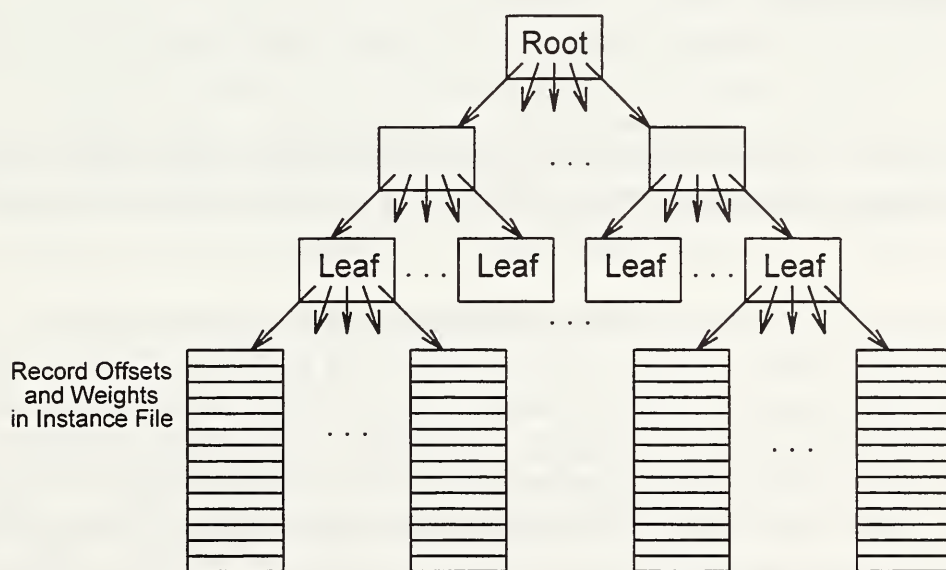


Figure 5-1 Format of Index & Instance files

A major factor in the length of time necessary to index a data file is the number of disk accesses. In order to minimize this number, the B+ tree is stored in memory during the indexing process as it will only grow to a reasonably small size (676 keys for a digram, 17576 keys for a trigram, etc.), and the linked lists of offsets/weights for each n-gram are maintained in memory until they

reach a predetermined size. Once this size limit is reached, the linked list is copied to an array and added to a temporary instance file of linked lists on the magnetic disk.

Once each record in the data file has been indexed, the instance file of linked lists is restructured to contain all of the offsets/weights for a given n-gram in contiguous order. The offset to the beginning of each n-gram queue is stored as the pointer value to the n-gram in the B+ tree, which is now written to a file.

### 5.3.3 Searching Methodology

One method of obtaining ranked output from a search is to first retrieve all matches to each n-gram, merge these matches to arrive at a rank for each record, and then sort the records by their rank. In a large file, however, a sort process can be very costly in terms of time. This searching algorithm eliminates the last sorting step by using a hash table to "sort" the records by their rank during the merge process.

During the search process, the name field is split into overlapping n-grams in the same manner as the indexing process. Each n-gram is searched in the B+-tree, returning an instance list for that n-gram. These lists are in descending order by record offset, and can be merged together to determine the total number of n-grams in common between the data record and the search record.

Once the number of common n-grams is known, a record can be given a rank. The matching function that was used to measure the similarity between a given record and the search query is the Dice Coefficient, which is

$$2c/(k + l)$$

where  $c$  is the number of n-grams in common,  $k$  and  $l$  are the number of n-grams in the search query and the data record, respectively (Perry & Willett, 1983). The rank, as calculated, is a real number between 0 and 1.

The matches are sorted by inserting them into a hash table, using the rank as the key. The hash function truncates the key after a certain number of digits of precision and then converts this value to an integer. For example, if the calculated rank is 0.8752, then the output of the hash function would be 87 when using two digits of precision, 875 when using three digits of precision, and 8752 when using four digits of precision. The number of digits of precision used is important because if it is too large then the sorting algorithm is not as efficient, and if it is too small, then the matches are not ranked well. Figure 5.2 shows the structure of the hash table which is generated by this step.

Once the queues have been merged, ranked, and sorted, the result is a table with a linked list of record offsets chained from the array index corresponding to their rank. At this point, each of the linked lists can be walked, beginning with the largest array index, to generate a ranked output of matches.

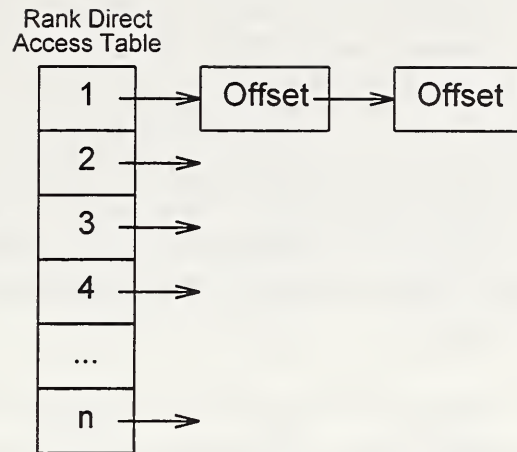


Figure 5-2 Structure of hash table

### 5.3.4 Pruning Methodology

The search algorithm will retrieve all of the records that match any part of the user's input query, even if there is only one n-gram that matches. Consequently, many of the matches that are returned are not of interest to a searcher, and the inclusion of those matches will slow down the retrieval process to a great degree. Pruning of records is done to remove records which are below a certain weight threshold, and thereby speeds up the retrieval process.

Pruning is also done by the frequency of occurrence of a certain n-gram within the file. This is based upon the assumption that if a particular n-gram is common enough, its inclusion will not contribute much to the rank of the records containing the n-gram. Also, it will bring in a large number of matches which will only be noise, and consequently slow down the retrieval process. The n-gram is then pruned before any matches are retrieved from it. This pruning should be done with caution, for if too many n-grams are pruned, or the frequency threshold is set too low, then the ranking of the relevant matches can be seriously affected.

## 5.4 Laboratory Results and Performance Evaluation

Experiments were run to determine the effectiveness of this methodology on the employer reports. The experiments used a subset of the data collection, consisting of 26 reports by United States employers of wages paid to employees, ranging in size from 10,000 to 500,000 records. From each report, 200 names were chosen at random. These names were reduced to a search name, emulating the type of name a user might enter into a search query. For example, if the name in the file was "Jones, Paul F.", the search name would be "Paul Jones". The search name was then modified to include common types of errors (Hall & Dowling, 1980). For each search name, a character was randomly inserted, deleted, substituted and reversed to yield another search name. Therefore, a total of 1000 searches (200 times 5 variations) were performed against each file, for a total of 26,000 searches.



For each search performed, in addition to the original record which is considered the relevant record, there were two types of similar records which could be found in the output. The first type of similar record would match in both the first and last names but may have other words in the name field, for example, "Jones, Paul E.", "Jones, Paul Stephen", "Jones-Wilkes, Paul", etc. The second type of similar record would be a close match in either or both of the names, possibly coupled with an exact match of one of the names. For example, "Jones, Gary", "Williams, Paul", "Janes, Paulie", etc. This second type of similar record was important to potential users because of the need to reconcile maiden names and parts of hyphenated names, and because of the high number of errors in the name fields.

Three methods of evaluation were used: the presence or absence of the original record in the output, the number of records ranked above the original record -- the Expected Search Length (ESL), and the number of similar records in the output. The last two methods were done to evaluate the usefulness of the system to potential users.

The number of similar records in the output was determined by computing the length of the Longest Common Subsequence (LCS) between the search name and each of the matches. Given two strings, X and Y, the LCS is defined as the longest subsequence common to both X and Y (Cormen, Leiserson, & Rivest, 1990). For example, for the names "Paul Jones" and "Peter Janes", the LCS would be <P,J,N,E,S>, or 5. An output record with a LCS length that was at least 80% of the search name length was considered to be a match of type one. An output record with a LCS length that was at least 50% of the search name length was considered to be a match of type two. For the above example, the LCS length of the output name is 55% of the search name length, and would be considered a type two match.

As soon as experimentation began, it was apparent that some kind of pruning would need to be done as many of the matches returned were only minorly similar, and the search times were not optimal. Until experiments with pruning were actually performed, all records which did not have a Dice Coefficient of at least 0.4 were pruned.

#### **5.4.1 N-gram Experiments**

The first set of experiments that were run consisted of varying the size of n, and the overlap of the n-grams. These experiments were performed on a smaller subset of the data, only 10 reports, as the tests involved indexing each of the reports multiple times. For each of the test combinations -- Digram overlapped (2G/OL), Digram non overlapped (2G/NO), Trigram overlapped (3G/OL) and Trigram non overlapped (3G/NO) -- the base name was searched, as well as a name containing an insertion, a deletion, a substitution, and a reversal. For each of the types of searches, the expected search length was calculated, and the number of similar records in the output was calculated. The results from these experiments are shown in Table 5-1.

Search Method	Search Type	#names searched	#names found	Expected Srch Len	Type 1 Similar (Avg.)	Type 2 Similar (Avg.)	Dissimilar (Avg.)
2G/OL	base	60	60	1.12	110.57	464.4	46.82
2G/OL	insert	119	119	1.37	42.98	386.73	37.17
2G/OL	delete	120	120	1.13	134.95	375.77	21.18
2G/OL	substitute	119	119	1.54	63.3	356.73	42.24
2G/OL	reverse	120	120	1.65	63.16	316.29	30.57
2G/NO	base	60	60	1.08	97.38	459.37	83.48
2G/NO	insert	120	109	16.83	27.93	273.78	57.68
2G/NO	delete	120	116	24.18	110.91	371.97	37.97
2G/NO	substitute	120	120	2.42	56.73	361.81	79.86
2G/NO	reverse	119	119	4.45	56.65	328.12	53.85
3G/OL	base	60	60	1.08	79.83	212.92	4.52
3G/OL	insert	120	120	1.45	30.6	187.3	5.53
3G/OL	delete	120	120	2.32	88.9	135.18	2.22
3G/OL	substitute	120	120	3.74	47.75	157.26	4.36
3G/OL	reverse	119	118	7.99	45.68	137.04	3.39
3G/NO	base	60	51	94.08	70.08	255.3	26.68
3G/NO	insert	120	99	71.08	20.74	203.97	40.63
3G/NO	delete	119	92	74.76	80	246.36	21.92
3G/NO	substitute	120	92	105.78	39.28	211.63	26.02
3G/NO	reverse	120	83	89.73	37.06	186.57	32.08

Table 5-1. N-gram Experiment Results

Based upon these results, it was determined that overlapping the n-grams was necessary, and that the digrams performed better than the trigrams. This can be seen in the chart, where for the overlapping digrams, all of the searched records were found, and in the non-overlapped digrams, 15 of the searched records were not found. For the overlapping trigrams, only one of the search records was not found, however for the non-overlapped trigrams, the number of records not found increased to 122.

The expected search length was the shortest using overlapping digrams, with an increase using overlapping trigrams. However, using non-overlapping digrams and trigrams, the expected search length increased dramatically.

The Table 5-2 shows the indexing time and index size for several different sizes of report files using overlapping digrams. The remainder of the experiments were run using overlapping digrams.



File Size (records)	Index Time (minutes)	Index Size (MB)
30186	1.5	2.57
123565	6.5	10.59
250001	17.6	27.20

Table 5-2. Index Time and Size (Elapsed Time)

### 5.4.2 Ranking Experiments

The Dice Coefficient was used to rank the records returned from a search in terms of their proximity to the search query. The matches are then sorted by rank by inserting them into a hash table, using the rank as the key. As the effectiveness of the ranking process is dependent on the number of digits of precision of the rank used in the hash function, experiments were performed using several different values: two, three and four. The results of the experiments for one data set are in Table 5-3.

Digits of Precision	Search Type	#names searched	#names found	Expected Srch Len	Type 1 Similar	Type 2 Similar (Avg.)	Dissimilar (Avg.)
2	base	200	200	2.26	173.11	784.40	44.83
	insert	200	200	2.18	57.91	545.23	54.31
	delete	200	199	2.23	172.53	575.05	25.63
	substitute	200	200	2.75	92.37	551.50	44.20
	reverse	200	200	3.50	79.02	463.70	30.77
3	base	200	200	1.48	173.11	784.40	44.83
	insert	200	200	1.48	57.91	545.23	54.31
	delete	200	199	1.65	172.53	575.05	25.63
	substitute	200	200	1.66	92.37	551.50	44.2
	reverse	200	200	2.96	79.02	463.70	30.77
4	base	200	200	1.45	173.11	784.40	44.83
	insert	200	200	1.48	57.91	545.23	54.31
	delete	200	199	1.64	172.53	575.05	25.63
	substitute	200	200	1.67	92.37	551.50	44.2
	reverse	200	200	2.93	79.02	463.70	30.77

Table 5-3. Scaling Experiments



By changing the digits of precision from two to three, the expected search length was decreased. In addition, the average search time using two digits of precision was slightly higher than the average search time using three digits of precision, 1.08 cpu seconds as opposed to 1.07 cpu seconds. When the digits of precision were increased to four, the expected search length did not decrease as much, and in some cases remained the same or increased. Furthermore, the average search time using four digits of precision increased slightly over that seen with three digits of precision, 1.14 cpu seconds as opposed to 1.07 cpu seconds. Note that the actual number of similar records found in the output remained constant irregardless of the digits of precision used. Based upon these results, three digits of precision were used in the pruning experiments.

### 5.4.3 Pruning Experiments

Pruning of records that fell below certain thresholds was done in order to optimize the search time. Two methods were tested -- pruning by the frequency of the gram in the file, and by the Dice Coefficient value of the record. Pruning by the frequency in the data file assumes that at some point, a given gram will occur in so many records that its inclusion only adds noise to the search process. It is necessary, however, to have some safeguard present so that enough of the n-grams remain to have a valid search. Pruning by the Dice Coefficient value assumes that a record with a rank below a given threshold is not a valid match. Table 5-4 shows the results of pruning by the Dice Coefficient value for one report. Results of the pruning by frequency of the gram in the file are being gathered.

Minimum Dice Value	Search Type	#names searched	#names found	Expected Srch Len	Type 1 Similar (Avg.)	Type 2 Similar (Avg.)	Dissimilar
0.4	base	200	200	1.48	173.11	784.40	44.83
	insert	200	200	1.48	57.91	545.23	54.31
	delete	200	199	1.65	172.53	575.05	25.63
	substitute	200	200	1.66	92.37	551.50	44.2
	reverse	200	200	2.96	79.02	463.70	30.77
0.3	base	200	200	1.48	229.81	2523.2	408.82
	insert	200	200	1.48	84.31	1758.3	465.01
	delete	200	200	9.36	279.80	2214.1	273.91
	substitute	200	200	1.66	133.11	1945.3	375.00
	reverse	200	200	2.96	123.72	1751	306.00

Table 5-4. Pruning by Dice Coefficient Results

When using a minimum Dice value of 0.4, one of the original search records was not found in the output, but when the minimum dice value was decreased to 0.3, the record was found. The expected search length did not change except in the case where the additional record was found, however the number of similar records found in the output increased substantially, as did the number of dissimilar records. The average search time increased slightly from pruning at 0.4 to pruning at 0.3, with 1.12 cpu seconds as opposed to 1.07 cpu seconds.

## 5.5 Areas of Further Research

There are several areas which are targeted for future research which may yield improvement in the results of the searching methodology. Some of these areas are explained below.

- The Social Security Number is now broken into three chunks - using the logical divisions of the number. For example, "212 33 4323" would be broken into "212", "33", and "4323". If the number were divided into true digrams or trigrams, it may yield an improvement in ranking. However, it may only serve as a detriment in the ranking as with the division of the number as it is, it forces a matching section of the SSN to be in the correct location (as each of the sections is different in length).
- When a word is broken into n-grams, only the n-gram combinations with actual letters are used. An improvement may be gained by adding additional n-gram combinations which signify the beginning and ending of a word. For example, the name "JOE" is now broken into "JO" and "OE". Adding additional combinations would yield "J", "JO", "OE", and "E".
- In the laboratory experiments that were conducted, it was apparent that the use of digrams gave better performance than the use of trigrams. However, a mix of digrams and trigrams may perform well, and may make searches on larger files faster.

## 6 Search Engine Design and Implementation

This section describes the actual implementation strategy for the search engine. It includes sections on the high level data structures used in the searching and indexing code, the defined constants and tunable parameters used to control the indexing and searching process, and the source code modules used in the implementation of the indexing, searching and data control processes.

### 6.1 High Level Data Structures

This section describes the high level data structures that were used to implement the search engine. It breaks up the structures into two types - visible structures, those that are common to both the search engine and the user interface, and invisible structures, those that are only used by the search engine.

#### Visible Structures (Structures Used Externally to the Search Engine)

##### EAMATE Data Structures

- Employer Header Record (struct EAMATE\_W2EMPLR\_HEADER)
- Employer Intermediate Total Record (struct EAMATE\_W2INTERMED\_TOT)
- Employer Final Total Record (struct EAMATE\_W2FINAL\_TOT)
- Employer Cumulative EIN Total Record (struct EAMATE\_W2CUMEIN\_TOT)
- Employer Header Information Record (struct EAMATE\_W2EMPLR\_INFO)
- Employee Detail Record (struct EAMATE\_W2EMPL\_DETAIL)
- Employee Browse Record (struct EAMATE\_W2EMPL\_BRW)

##### Communication Data Structures

- User Query Structure (struct USER\_QUERY)

#### Invisible Structures (Structures Used Internally to the Search Engine)

##### B+ Tree Data Structures

- B+ Tree node as stored in memory (struct MEMNODE)
  - count : Number of Keys in the Node
  - key : Array of Keys in the Node
  - leaf : Boolean Value Indicating whether node is a leaf node
  - self\_offset : Offset of this node in the data file (on disk)
  - num\_dupe : Count of duplicate records for each key
  - dupe\_offset: Offset to Duplicate list in temporary posting file
  - branch : union of data pointers to memory dupe offsets (leaf node), memory to child nodes (non-leaf node), or offset to either on the disk



- B+ Tree node as stored on disk (struct NODE)
  - count : Number of Keys in the Node
  - key : Array of Keys in the Node
  - freq : Count of duplicate records for each key
  - leaf : Boolean Value Indicating whether node is a leaf node
  - branch : disk offset to child or duplicate list
- Duplicate Linked Lists as stored in memory (struct KEYLIST)
  - offset : Disk offset to Record in Browse File
  - weight : Weight of Record (number of grams)
  - next\_key : Pointer to next key in linked list
- Duplicate Lists as stored on disk in final file (struct RECINFO)
  - dupe\_offset : Disk offset to the record in the browse file
  - dupe\_weight : Weight of Record (number of grams)
- Duplicate Lists as stored on disk in temp file (struct DUPELIST)
  - record : Array of MAXDUPE structures of RECINFO
  - next\_set : Disk offset to next set of records

### **Employer Index Data Structure**

- Employer index key (struct EMPLR\_IDX)
  - EIN : Employer Identification Number
  - offset : Offset to Employer Header Information Structure in Employer Header File

### **Control and Statistics Data Structures**

- Parse Control Data Structure (struct CTRL\_FILE)
  - EIN : Employer Identification Number
  - seq : Latest sequence number used by this EIN
  - browseloc : Location of the Browse file for this EIN
- Statistics Gathering Structure for Search (struct TEST\_DATA)
  - user : The user number of the searcher
  - year : Year of the Data
  - EIN : EIN being searched
  - first : First name entered by the user
  - last : Last name entered by the user
  - searchname: Actual search name
  - SSN : SSN entered by the user
  - numgrams : Number of n-grams used in search
  - nummatch : Number of matches returned by the search

- recspruned: Number of matches pruned by the search
- grpruned : Number of n-grams pruned by the search
- etime : Elapsed time of the search
- wtime : Write time of the search
- mem : Memory Allocated by the search
- ctime : CPU time used by the search

## 6.2 Tunable Parameters

The following are parameters used to control the indexing and searching process. The values indicated were the ones that were determined to be the most efficient for our prototype system. This may or may not carry through to a larger system. Where possible, differences have been noted. These parameters are listed in the params.h file. Other parameters besides the ones listed below were in the parameters header file, however these were just size definitions for arrays and such, and were not critical to the system performance, therefore they are not explained here.

### Index Parameters

```
MINCHILD    : 310
MAXCHILD    : 2*MINCHILD-1
```

These parameters are used in the creation and maintenance of the B+ tree. They define the minimum and maximum number of children a B+ tree node may have (or the minimum/maximum number of keys). These parameters also have significance during the search process.

```
GRAM_SIZE   : 2
```

This is the size of n-gram that the names have been divided into. We found that digrams worked better than trigrams for our system, however it may be possible that a mixture of digrams and trigrams would work well.

```
OVERLAP     : TRUE
```

For the prototype, overlapping grams were used, rather than the non-overlapping variety. We found this to be much more robust in detecting errors, and almost essential in the case of insertions and deletions of characters.

```
MAXGRAMS    : 26*26
```

This is the maximum number of n-gram combinations possible in the names. For the names, we were stripping the data to only alpha characters, which yielded the alphabet squared. Obviously, if we were to extend to allow other characters, the maximum number of grams would increase.

However, many of the characters could be construed to be errors that a user would not think to input, so they were not considered.

MAX\_GRAM\_SIZE : 4

This is the maximum gram size to be used in the system. This number is important because SSNs are parsed differently than the names -- in the sections by which they naturally divide. Therefore, a gram could potentially be 4 characters long for an SSN. This is necessary for allocating space for index files.

SEQ\_SIZE : 3

This is the sequence length for use in numbering multiple employer reports, which is a character string from AAA-ZZZ. This sequence size could be increased if it were determined that a single employer had more than 17576 reports for a given year (including AWR and corrected reports).

DUPEFILE : TRUE

This parameter determines whether records with the same gram are stored in a duplicate postings list, or in the B+ tree itself. The duplicate file is used because the overhead associated with storing each gram in the B+ tree is astronomical, and the storage space needed to store the index data is decreased by several orders of magnitude by the use of the duplicate file. It also makes merging the queues of matches very efficient during the search process.

MAXDUPE : 512

This parameter is used to determine the number of duplicate matches for each gram which are stored in memory before they are written to disk during the indexing process. This is a very important parameter to scale properly, as it controls the efficiency of the index process to a large degree. If the number is too low, causing many more writes to the disk, then the index process can be drastically slowed. If the number is too large, and there is not enough memory on the system, then disk thrashing can be caused as the system encounters a large number of page faults as large sections of memory are swapped to disk.

### **Search Parameters**

DICE : TRUE

This parameter is used to determine whether to use the Dice Coefficient in calculating the weight of a match with regard to the search query. If this is set to FALSE, then the straight count of grams that are matching between the search and the record are used to determine weight. This allows no normalization for the length of the record, and the use of the Dice Coefficient is recommended.



DICE\_SCALE : 100

This parameter is used to scale the dice weight to an integer value. The actual dice weight is a number between 0 and 1. It is necessary to scale this number to an integer value due to the linear sort routine used in the search process. The value of the scale is important because it will affect the overall ranking used in the search process because significant digits are truncated during the conversion, the number of which is determined by the scale.

DICE\_GRAIN : 150

This parameter is used to determine the range of possible values for the dice weight. It is used as a space allocator for the search process. It should be set to 1.5 times the dice scale.

PRUNE : TRUE

This parameter is used to set on or off the pruning based upon the dice value. It is significant because the search engine will pick up records that match only one gram. A low gram may not be a significant match, and the entire processing of large quantities of these matches (of which there will be many) can significantly slow down the performance of the search engine.

PRUNE\_WEIGHT : 33

This parameter determines the dice weight at which to prune records as an inconsequential match.

FPRUNE : TRUE

This parameter turns on or off the parameter which allows for pruning due to the frequency of a particular gram in the file. It is based upon the theory that if a particular gram is present in a file to such a degree that it is considered "common", it will not have any significance in the ranking of matches.

FPRUNE\_LEVEL : 0.2

This parameter will set the level at which a gram is considered to be too prevalent to have any significance in the match. The value is actually a percent, with 1.0 being 100%.

MIN\_GRAMS : 5

This parameter will set the minimum number of grams that will be allowed in a search. This will prevent the frequency pruning from totally pruning away the search query because it may be a very common sequence.

INITIALSCALE : 1000  
INITIALONLY : 1.2  
INITIALFACTOR : 2

These parameters are used to determine whether an employer report contains full names or initials in the employee name field. They are based upon the word count in the name field. At this point they are not really used, but an initials value is stored in the employer header information.

REALLOCFACTOR : 10

This parameter is used when memory is reallocated. It determines the number of "chunks" of memory that are to be reallocated. A "chunk" is some kind of data structure, i.e., long word, structure type, etc.

### **User-Centered Search Parameters**

BLANKETSIZE : 30  
BLANKETFACTOR : 10

These parameters are used in the blanket request query. BLANKETSIZE determines the number of records that should be pulled from the report in total. BLANKETFACTOR determines the number of records that should be pulled from the beginning, the middle and the end of the report.

BROWSENUM : 1000

This parameter is used to determine how many records should be sent to the user during a single query or browse report session. This number is determined by the memory capacity of the workstation and by the estimation of the number of records viewed before the user either finds the match or determines that there is no match.

SET\_SIZE : 50

This parameter is used to determine how many browse offsets returned from a search should be converted to actual browse data. The parameter should not be too low, or the user interface will continue to request additional matches. However, if the parameter is set too high, then the search can take much longer, and there is a high probability that the user will not even need to look at so many matches.

### **6.3 Source Code Modules**

The index and search code is divided logically into groups of code that fulfill specific purposes. The groups are further divided into source code modules that reflect their purpose. The source

code is stored in a project hierarchy which uses the UNIX utility "make" to control the maintenance of the source code. The hierarchy, with the parent directory being the home directory for the user account, is as follows (Directories with an "\*" have a Makefile associated with that directory.):

Project Parent Directory*:	ssacode
Include Files:	ssacode/include
Library Files:	ssacode/lib
Executable Files:	ssacode/bin
Source Files*:	ssacode/src
Library Source Files*:	ssacode/src/lib
Application Source Files*:	ssacode/src/bin

In the Library Source Files directory and the Application Source Files directory, there is a sub directory for each source code module. Each of these directories has a Makefile associated with it to control the maintenance of the modules. A complete listing of the source code and the Makefiles is in Appendix D. The source code modules and the directories in which they reside, and the include files used are as follows:

## Include Files

ssacode/include/btreestruct.h

This include file contains defines and structure definitions for the creation and maintenance of the B+ trees.

ssacode/include/eamatestruct.h

This include file contains defines and structure definitions specific to the EAMATE data

ssacode/include/params.h

This include file contains defines which are used to configure and control the indexing and searching process for the EAMATE data

ssacode/include/paramsemplr.h

This include file contains defines which are used to configure and control the indexing process for the employer header information.

## Library Modules

ssacode/src/lib/btree\_data/btree\_data.c                   -- archived in libbtree\_data.a

This source code module contains functions used to create and maintain the btree structure used in indexing and searching the EAMATE data. It is the same as "btree\_emplr.c", except that it includes "params.h", instead of "paramsemplr.h".

ssacode/src/lib/btree\_emplr/btree\_emplr.c               -- archived in libbtree\_emplr.a

This source code module contains functions used to create and maintain the btree structure used in indexing of the employer header information. It is the same as "btree\_data.c", except that it includes "paramsemplr.h", instead of "params.h".



ssacode/src/lib/general/eamate.c -- archived in libgen\_eamate.a  
 This source code module contains functions that are specific to the handling of the EAMATE data.

ssacode/src/lib/general/general.c -- archived in libgen\_eamate.a  
 This source code module contains functions that are general to many application modules.

ssacode/src/lib/test/test\_funcs.c -- archived in libtest.a  
 This source code module contains functions that are used to control and view data. They provide a wide range of debugging capabilities.

## Application Modules

ssacode/src/bin/client/client.c -- compiled to client  
 This source code module is a generic client used to exercise each option of the search engine. It allows the input of a user query, and sends it to the search engine.

ssacode/src/bin/debug/debug.c -- compiled to debug  
 This source code module is used for debugging purposes. It allows a user to view and manipulate many of the system and control files generated by the search engine.

ssacode/src/bin/download/download.c -- compiled to download  
 This source code module is used to download the tapes containing the EAMATE data.

ssacode/src/bin/fix\_parse/fix\_parse.c -- compiled to fix\_parse  
 This source code module is used to remove an employer (by sequence number or in its entirety) from the parsed employers.

ssacode/src/bin/index/index.c -- compiled to index  
 This source code module is used to index an EAMATE employer file (once it has been parsed).

ssacode/src/bin/indexemplr/indexemplr.c -- compiled to indexemplr  
 This source code module is used to index the employer header information file.

ssacode/src/bin/parse/parse.c -- compiled to parse  
 This source code module is used to parse an EAMATE employer file from the COM format (as delivered by SSA) into the format used by the indexing routine/search engine.

ssacode/src/bin/search\_addmatch/search\_addmatch.c -- compiled to search\_addmatch  
 This source code module is part of the actual search engine. It contains RPC functions to add matches to a browse list, starting at a particular offset.

ssacode/src/bin/search\_blanket/search\_blanket.c -- compiled to search\_blanket  
 This source code module is part of the actual search engine. It contains RPC functions to create a potential blanket report from an employer report.

ssacode/src/bin/search\_browse/search\_browse.c -- compiled to search\_browse  
 This source code module is part of the actual search engine. It contains RPC functions to create a browse file from an employer report, containing records for a user to browse.

ssacode/src/bin/search\_detail/search\_detail.c -- compiled to search\_detail  
 This source code module is part of the actual search engine. It contains RPC functions to retrieve a detailed employee record from a particular employer report, at a particular offset.

ssacode/src/bin/search\_header/search\_header.c                   -- compiled to search\_header  
This source code module is part of the actual search engine. It contains RPC functions to search for employer header information for a particular employer report.

ssacode/src/bin/search\_print/search\_print.c                   -- compiled to search\_print  
This source code module is part of the actual search engine. It contains RPC functions to print an employer report.

ssacode/src/bin/search\_single/search\_single.c               -- compiled to search\_single  
This source code module is part of the actual search engine. It contains RPC functions to search for a single person in a particular employer report.

ssacode/src/bin/sysadm\_print/sysadm\_print.c               -- compiled to sysadm\_print  
This source code module is used by the system administrator in order to print large employer report files. It is separate from the search\_print application in order to limit the number of people who can print large amounts of data.





## 7 Conversion and Indexing of the EAMATE Data

The EAMATE data, as received from SSA, is stored on 1/2" open reel tapes. The data is in the format used to print to microfilm, containing white space, data labels, and COM characters. It is necessary to parse the data out of the employer reports, and to index the data. In addition, checks are performed during this process in order to verify the integrity of the data and to gather data statistics. This section describes the process of converting and indexing the EAMATE data, and the applications used to do this.

### 7.1 Directory and File Structure Required

When the data is converted and indexed, it will be stored in a certain format on the file server. The file hierarchy starts at the root level of a user account, for our purposes, it will be called "ssa". This is the same account which the client workstations will log into at system startup. The directory hierarchy is as follows:

Root directory of the user account:	/export/home/ssa
Browse files of employer reports:	/export/home/ssa/1991BRW/1
	...
	/export/home/ssa/1991BRW/n
Detail files of employer reports:	/export/home/ssa/1991DET/1
	...
	/export/home/ssa/1991DET/n
Directory to contain unparsed reports:	/export/home/ssa/datahold

In each of these directories, there are files that are used to convert, index and search the EAMATE reports. Some of these files are created by the system administrator in order to process the data, and other files are created by the programs as the data is converted and indexed. The following subsections describe the files which will be placed in each of the sub directories, and which are necessary for the indexing and searching of the data. In sections 7.2 and 7.3, files will be described which facilitate the data conversion and indexing, or are byproducts of one of the three applications, but which may be moved at a later date.

### Files in the Root Directory of the User Account

In the root directory of the account, several types of files will be stored. The following is a list of all of the files, and a short explanation of what is contained in the file.

### **1991.employers.text**

A file of structures of type EAMATE\_W2EMPLR\_INFO, one for each report that has been parsed.

### **1991.employers.idx**

A file of structures of type EMPLR\_IDX, which is a sorted index to the 1991.employers.text file. There is one record in the file for each record in the 1991.employers.txt file.

### **1991.<EIN>.name.idx**

### **1991.<EIN>.ssn.idx**

These files are B+ tree files, containing structures of type NODE, which serve as an index to the duplicate postings file. There is one node for each queue in the duplicate postings file, which points to the start of the queue. There is one file of each type for each EIN for the year (It is important to point out that this is NOT the same as one for each report submitted for the year). The individual reports for an employer are indexed together due to the need to search all reports for a given employee during a "Single Query" request. <EIN> is an employer identification number, in the format of XX-XXXXXXX (example: 75-1232789). If there is not enough space in the disk partition which is mounted to the root of the home account, these files can be stored elsewhere, but the files **MUST** be symbolically linked back to this directory.

### **1991.<EIN>.name.dup**

### **1991.<EIN>.ssn.dup**

These files are duplicate postings files, containing structures of type RECINFO, which are organized into queues. There is one such queue for each n-gram combination, which contains record offsets and weights to each record in an employer report which contains the n-gram. There is one file of each type for each EIN for the year (It is important to point out that this is NOT the same as one for each report submitted for the year). The individual reports for an employer are indexed together due to the need to search all reports for a given employee during a "Single Query" request. <EIN> is an employer identification number, in the format of XX-XXXXXXX (example: 75-1232789). If there is not enough space in the disk partition which is mounted to the root of the home account, these files can be stored elsewhere, but the files **MUST** be symbolically linked back to this directory.

## **Application Programs**

The application programs needed to convert, index and search the data are also stored in this directory. Only the executable versions are stored here, the source code is stored in the development directory in the project hierarchy as outlined in section 6.3.

### **Files contained in the Browse Subdirectories**

The 1991BRW directory is where the browse data files are stored for the employer reports. There is a series of sub directories beneath this directory called 1, 2, 3 .. n, which are mount points for the magnetic hard disks which will be used to store the browse data files. There should be one such sub directory for each of the magnetic hard drives in the system, and it is important that the same magnetic hard drive is mounted to it at each system startup. The browse data is physically stored in each of the sub directories, and symbolically linked to the directory above (the 1991BRW directory) using the "ln" command (example: `ln -s <#>/* .`, where <#> is the sub directory number, and the command is executed from the 1991BRW directory).

The browse files are a file of structures EAMATE\_W2EMPL\_BRW. There is one browse file for each EIN, containing the browse data for each record in all of the reports filed by that employer. The browse records are all stored together in one file because of the need to search all reports for a given employee during a "single query" request. However, each report can be individually browsed, printed and requested for blanket information. The files are named XXXXXXXXX.X, where X is the EIN (example: 75123278.9).

### **Files contained in the Detail Subdirectories**

The 1991DET directory is where the detail data files are stored for the employer reports. There is a series of sub directories beneath this directory called 1, 2, 3 .. n, which are mount points for the optical disk platters which will be used to store the detail data files. There should be one such sub directory for each side of each optical disk platter used in the system, and it is important that the same platter/side is mounted to it at each system startup. The detail data is physically stored in each of the sub directories.

The detail files begin with a structure of type EAMATE\_W2EMPLR\_HEADER, with the body containing a mixture of structures of type EAMATE\_W2EMPL\_DETAIL and EAMATE\_W2INTERMED\_TOT, and end with a structure of type EAMATE\_W2FINAL\_TOT. There is one employer detail file for each employer report that was parsed, and they are named XXXXXXXXX.XYYY, where X is the EIN and Y is a sequence number from AAA-ZZZ. This gives a maximum number of reports per employer of 17576.



## **Files contained in the Datahold subdirectory**

The datahold sub directory is used to store the employer reports in COM format before they are parsed, and placed in the 1991BRW and 1991DET directories. Once the files are parsed, the original files can be backed up to tape and deleted, as they are no longer used by the file server applications.

Also stored in this directory is the "download" application which is used to download the unconverted COM files from 1/2" open reel tape to the system hard drive.

## **7.2 Walkthru of the Data Conversion Process**

The EAMATE data goes through several conversion and verification steps to progress from the COM format received from SSA to the indexed format used by the search application. This section describes the steps and applications used to convert the data to an indexable form, and to verify that no errors occurred during the conversion. In addition, the files that are required in order to convert the data, and which are produced as byproducts of the conversion are described. Section 7.3 describes the steps and data files involved in indexing the data. (NOTE: For disk space requirements, it may work better to parse the large employer reports first, and then continue with the smaller employer reports.)

1. As the data was delivered on 1/2" open reel tape, it was first necessary to pull the data from the tape and store it in the "datahold" sub directory. The application used for this is "download". (NOTE: The code for "download.c" may have to be changed for the device driver name which is on the Sun system being used. At this point, it is set to use /dev/rmt/0).

From the "datahold" sub directory, the application is run by typing "download" at the prompt. The data from the tape will be copied to a file called "employer.xxx", where "xxx" is a sequential integer number. When the application is started, the user is prompted to enter the initial number for "xxx", and this number will be automatically incremented with each tape that is downloaded. The data from each tape is stored in a separate file, and when entering the counter, the user should take care not to enter a number which will overwrite an existing data file.

A special case occurs when a single employer report spans more than one tape. Each tape would be downloaded to its own employer file, but it is VITAL that the files then be merged together in the proper order to form one file containing the entire report. The component files should then be deleted. If this is not done, the next step in the conversion will fail. The files can be merged using the "cat" command, and redirecting the output to a new file.

2. Once the data is in the datahold sub directory, it can be parsed. This step will strip the white space, data labels and COM characters from the employer file, and will store the data in the browse and detail formats. In addition, it creates and/or updates the 1991.employers.text file, the parse control file, and the parse statistics file. Other byproducts of this step are two files containing all of the names and SSNs in the report.

**This step is the one step where it is VITAL to be accurate during the entire process.** If a data file is parsed incorrectly, it requires restructuring the 1991.employers.text files and the parse control files, and possibly the browse file for that EIN. As it is imperative that these files be absolutely correct for the system to function properly, reconstructing the files can be a painstaking effort.

The application used to parse the data is the "parse" program. This program is run from the home directory of the account, and requires the presence of the einlist.parse file in the same directory. The einlist.parse file is created by the person running the application, and contains a list of employer files to parse, the detail location and the browse location (see sample "einlist.parse" file in this section for the required format -- a redirection of the "ls" command can be used to create this file, and then the file can be modified in the "vi" editor to add the browse and detail locations). **When choosing the browse and detail locations, it is important to make sure that the files will fit in the partition to which they are being assigned. Also, if a previous report for an employer has been parsed, it should be kept in mind that all of the browse files for that employer will be written to that location (appended to that file).** The charts in Appendix F will help to determine the size that a browse file will be when compared to the size of the COM file.

To run the application, type "parse" at the prompt. The user will be prompted for the year of the data, and each file in the einlist.parse will be parsed and the debugging output will be placed in the parse.debug file. This file will contain several types of messages. It will contain a message for each report which is parsed, mapping the COM file name to an EIN and sequence number. It will contain a message when a previous report for the EIN has been parsed and will indicate where the browse location has been assigned (the browse location will ALWAYS be set to the browse location of the first employer report processed for this EIN, irregardless of what is specified in the einlist.parse file). These messages are not really errors, so they will not be prefaced with the "ERROR" flag. All other messages are considered errors, and will be prefaced with the string "ERROR". These types of messages are things such as a file not beginning with an employer header record, and an unidentified record type being found in the report. These are indications that the file may be corrupted. There are other errors which occur when a file cannot be opened, which should not occur if the permissions in the directory are proper. If any of these error messages are encountered, it will most likely be necessary to run the "fix\_parse" program.



Once the process is complete, several verification steps need to be taken. The first is to look at the `parse.debug` file for any errors which may have occurred. If this file is particularly large, the use of the UNIX utility `"grep"` can be used to look for all occurrences of the string `"ERROR"`. The next step is to use the `"debug"` application to verify the detail files. This can be done by the option `"Check Detail Files for All Data"`, and will verify that each detail file begins with an employer header record, and contains employee records in the proper location and ends with a final total record.

There is an option in the `"debug"` application called `"Display Unrecognized Character in Parse File"` which can be used to view the area of a file that is corrupted if this error occurs. There are also options in the application which can be used to view the browse data file (from the beginning or from an offset), the detail data file, the employer header information file, and the parse control file.

3. After the data is parsed and checked, then the browse links can be created between the browse data in the sub directories and the parent directory. This is done via the symbolic link command issued from the 1991BRW directory (example: `ln -s ./1/* .`). A link command should be issued for each sub directory of the 1991BRW directory, so that there is a symbolic link for each browse file in the 1991BRW directory.
4. Back up to tape the files containing all of the names and SSNs in the employer reports (`<XX-XXXXXXXX>.names` and `<XX-XXXXXXXX>.ssns`) and the `"parse.debug"` file, and erase the files from the root directory of the account. Also back up to tape the `"parsectrl"` file and the `"einstats"` file. They can be backed up using the `"tar"` UNIX utility.

This completes the data conversion process. Once the data is converted, it can be indexed in order to be ready to be searched. Detailed information on the indexing process can be found in section 7.3.

## **Files Created by or for the Data Conversion Process**

### **`einlist.parse`**

A list of employer reports to parse, containing the full path name of the employer report, an integer number indicating the detail location and an integer number indicating the browse location. This file is created by the user before running the `"parse"` application. (Column headings are added to the following example for clarity, but will not appear in the file.)



Employer Report Path Name	Detail	Browse
	Loc	Loc
/export/home/ssa/datahold/employer.0	1	1
/export/home/ssa/datahold/employer.21	1	1
/export/home/ssa/datahold/employer.213	1	2
/export/home/ssa/datahold/employer.234	1	2
/export/home/ssa/datahold/employer.243	2	1
/export/home/ssa/datahold/employer.249	2	2
/export/home/ssa/datahold/employer.25	2	2

### Sample "einlist.parse" file

#### **einstats**

A file containing statistics on the employer reports that have been parsed. This file contains the following information: EIN, Sequence number of the report, Number of records in the report, Word count of total number of words in the name field for the entire report, Average word count per record of words in the name field, Size (in KB) of the report after parse, Size (in KB) of the report before parse, Ratio of size after parse to size before parse, Size (in KB) of browse file, Ratio of browse size to pre-parse size, Browse location, and Detail location.

EIN	SEQ	Num Recs	Word Count	Avg WC	Txt Size	COM Size	Txt/ COM	Brw Size	Brw/ COM	BLoc	DLoc
39-1390489	AAA	29	59	2.034	8.64	15.33	0.564	2.83	0.185	1	1
38-2113393	AAA	2018	4080	2.022	560.73	967.89	0.579	197.07	0.204	1	1
38-2113393	AAB	50	104	2.080	14.59	26.50	0.551	4.88	0.184	1	1
38-2113393	AAC	1453	2938	2.022	403.88	696.82	0.580	141.89	0.204	1	1
38-2113393	AAD	163	330	2.025	45.76	79.10	0.579	15.92	0.201	1	1

### Sample "einstats" file

#### **parsectrl**

A file of structures of type PARSECTRL, which contains information on each EIN which has been parsed. It will keep track of the last sequence number assigned to a report of this EIN, and the browse location used for this EIN. This file is not necessary once all of the reports have been parsed, but is **VITAL** during the parse process, therefore DO NOT erase this file during the parse process. It is also a good log of the number of reports per EIN which were processed, and the number of EINs that were processed for that year, and will have value during the scaling up for a full production system.

**<XX-XXXXXXX>.names**  
**<XX-XXXXXXX>.ssns**

These files contain a list of names or SSNs that are contained in the files being parsed. <XX-XXXXXXX> is the EIN of the employer report (example: 75-1232789). These files are generated by the "parse" application, and can be copied to tape and removed from the directory that they were placed in upon creation. It is important not to erase these files from the directory until all reports for a given EIN have been parsed, because they are appended to as new reports are parsed. In addition, it is VITAL to save these files to tape, because they will be very useful in evaluating the effectiveness of the search engine in order to recreate and tune problem searches. These files should be moved out of the root directory of the account before the indexing process is started, as they will take up unnecessary space in that directory which will be needed for the index files.

### **parse.debug**

This file contains a list of all of the errors that occur during the parse process and a log of the map of COM file to the employer EIN and sequence number. It is not necessary to keep this file after the parse process is complete, and it may even be deleted periodically during the parsing of different reports. However, before deleting the file, make sure that it is backed up to tape (see instruction 4).

## **7.3 Walkthru of the Indexing Process**

This section describes the steps and applications used to index the data. In addition, the files that are required in order to index the data, and which are produced as by products of the indexing are described. Note: For disk space requirements, it may work better to index the large employer files first, and then continue with the smaller employer reports.

1. Once the data is parsed, it is indexed in order to allow the search algorithms to be time-efficient. This step will create an index to the browse data files which can be searched by the search application. The index files created during this step consist of:

1991.<XX-XXXXXXX>.name.idx  
1991.<XX-XXXXXXX>.name.dup  
1991.<XX-XXXXXXX>.ssn.idx  
1991.<XX-XXXXXXX>.ssn.dup

where <XX-XXXXXXX> indicates the EIN. In addition, it creates and/or updates the index statistics file.

The application used in order to accomplish this is the "index" application. This program is run from the home directory of the account, and **requires** the presence of the einlist.index file in the same directory. The einlist.index file is created by the user, and

contains a list of browse files to index, without any explicit path name (see the sample "einlist.index" file in this section for the required format -- a redirection of the "ls" command can be used to create this file).

To run the application, type "index" at the prompt. The user will be prompted for the year of the data. Type "1991". Each file in the einlist.index will be indexed. If the disk partition mounted to the root directory of the user account fills up before all files are indexed, the .idx and the .dup files can be stored on another partition, however, they **MUST** be symbolically linked back to the root directory of the user account.

Once the process is complete, several verification steps need to be taken. The first is to check to make sure that .idx and .dup files exist for each of the EINs in the einlist.index file. Also, check to make sure that no EINs have a .tmp file. This is an indication that an error occurred in the indexing of this file, and the file should be reindexed. Any file can be reindexed with impunity as there are no control files which are keeping status information for use at a later time, as in the parse step.

There are several options in the "debug" application which can be used to view files created in this step. This options include: "Merge and Display Btree and Dupe File", "Display a Btree File", and "Display a Duplicate Postings File".

2. After all of the data is indexed, it is necessary to index the employer header information file. This step will create the 1991.employers.idx file, which is an index (by EIN and sequence number) to the 1991.employers.text file. The application used to accomplish this is the "indexemplr" program. This application is run from the home directory of the account, and requires the presence of the 1991.employers.text file in the same directory (this file is created by the "parse" application).

To run the application, type "indexemplr" at the prompt. The 1991.employers.text file will be indexed, creating the 1991.employers.idx file.

There is an option in the "debug" application which can be used to view the file created in this step, the 1991.employers.idx file. This option is called "Display an Employer Index File".

This completes the data indexing process. Once the data is indexed, it can be searched. Detailed information on setting up the search process can be found in section 8.



## Files Created by or for the Indexing Process

### **einlist.index**

A list of browse filenames to index (by EIN), containing only the browse file name as it appears in the 1991BRW directory (no explicit path name should be given). This file is created by the user before running the "index" application.

```
23257815.2
31111231.5
34999000.0
38211339.3
39139048.9
53020461.6
58096428.6
```

Sample "einlist.index" file

### **indexstats**

A file containing statistics on the employer reports that are indexed. This file contains the following statistics: EIN, Number of records indexed, CPU time used for indexing, Elapsed time during the indexing process, Temporary magnetic hard disk space used during indexing, Size of name index file (in KB), Size of name duplicate file (in KB), size of SSN index file (in KB), Size of SSN duplicate file (in KB).

EIN	Num Recs	CPU Time	Elapsed Time	Temp Space	Name Idx	Name Dup	SSN Idx	SSN Dup
23-2578152	999	2.33	2	0.00	7.87	54.02	31.48	23.41
31-1112315	250	0.42	0	0.00	7.87	14.64	7.87	5.86
34-9990000	250001	557.66	688	29364.65	23.61	27814.53	94.45	5859.40
38-2113393	26779	52.20	66	1553.52	7.87	1954.83	188.91	627.63
39-1390489	29	0.07	1	0.00	7.87	2.61	7.87	0.68
53-0204616	1003	2.53	3	0.00	7.87	56.86	39.36	23.51
58-0964286	3995	8.65	10	20.02	7.87	220.23	70.84	93.63

Sample "indexstats" file

## 7.4 Overview of the Verification Programs

There are several applications on the file server that were written to verify the files created by the conversion, indexing and/or searching applications, and to debug those applications. This section describes those applications.

### Client

This application is a small user interface that was written for the Sun to exercise each of the options provided by the searching application. It will request a query from the user and will send the request to the search application.

### Debug

This application consists of many options used to verify the integrity of the files created by the conversion, indexing and searching applications. Below is a list of each of the options, and a short description of their function.

- *Check Detail Files for All Data*

This is a verification program for the Detail files. It will check an employer detail file to make sure that it begins with an employer header record, and contains only employee detail and intermediate total records, and ends with an employer final total record. The user is asked to enter the filename of the file containing the list of all of the detail filenames to be checked. This file can be created by using a redirected "ls" command in approximately the same manner that the einlist.index file was created. The only difference between the format of this file and the einlist.index file is that the files in this list should contain explicit path names.

- *Merge and Display Btree and Dupe File*

This program will merge and print a btree file and a duplicate postings file. This provides verification that the indexing program is working properly.

- *Display Unrecognized Character in Parse File*

This program will print the area in the COM file that is suspected to be corrupted. The user will be prompted for an offset, and the area before, including and after the offset will be printed.

- *Display Browse Data File*

This program will print the records in the browse data file, with a descriptive tag before each field value explaining the data in that field.

- *Display Browse Data File from an Offset*

This program will print the records in the browse data file, starting at a given offset, with a descriptive tag before each field value explaining the data in that field.

- *Display a Btree File*

This program will print the records in the btree file, with a descriptive tag before each field value explaining the data in that field.

- *Display a Detail File*

This program will print the records in the detail data file, with a header tag telling the type of record, and a descriptive tag before each field value explaining the data in that field.

- *Display a Duplicate Postings File*

This program will print the list of offsets in the duplicate postings file. This is not used that much as it is much more informative to print a merged btree and duplicate postings file.

- *Display an Employee Detail Record File*

This program will print a file of employee detail records, with a descriptive tag before each field value explaining the data in that field. This is used mostly to print the record in the detail<user#>.txt file, which is generated as a result of a detail request.

- *Display an Employer Header Info File*

This program will print the records in an employer header information file, with a descriptive tag before each field value explaining the data in that field.

- *Display an Employer Index File*

This program will print the records in an employer header information index file, with a descriptive tag before each field value explaining the data in that field.



- *Check a Search Stats File for Errors*

This program will check each record in the search statistics file, printing a record if the number of matches field is -1, indicating an error. The record data will be preceded with a descriptive tag explaining the data in that field.

- *Display an offset List File*

This program will print the records in the list of offsets file which is generated by a single query search request.

- *Display a Parse Control File*

This program will print the records in the parse control file, with a descriptive tag before each field value explaining the data in that field.

- *Display a Search Statistics File*

This program will print the records in a search statistics file, with a descriptive tag before each field value explaining the data in that field.

- *Make a Search Statistics Table*

This program will print the records in a search statistics file in a tabular format, with one record per line, and a descriptive tag at each column head explaining the data in that field.

- *Display a List of Browse Data from a List of Offsets*

This program will print a list of records from a browse data file, determining the records to print and the order in which they should be printed from a list of offsets file that is specified. Each record will have a descriptive tag before each field value explaining the data in that field.

- *Display a Query File*

This program will print the record in the user query file, with a descriptive tag before each field value explaining the data in that field.

## **Sysadm\_print**

This application is provided to allow the system administrator to print a large employer report if it is determined necessary. The prototype application "search\_print" will return an error if a user attempts to print a file which has greater than 5000 employees in it. This is done to place a small measure of security on the printing functions. The "sysadm\_print" application will request the full name of the employer report to be printed, and will format the file and spool the file to the printer. The file name should contain the entire path to the file, relative to the current location. For example: 1991DET/1/39139048.9AAA would be the filename when the program is invoked from the home directory of the "ssa" account. The system administrator will then be requested to remove the temporary format file which is created. It is not removed automatically because the spooler uses the file long after the application finishes, and the file should not be removed until it is done printing.

## **Fix\_parse**

This program is provided to allow the system administrator to remove an EIN (either by sequence number or in its entirety) from the parse control files. This application can be used if the parse process has encountered an error which has corrupted the parse control files. (NOTE: The "fix parse by sequence number" choice of the program should only be chosen if a report will still exist for that EIN.) This program will modify all of the files which cannot be directly removed, and will provide the user with a prompt telling which of the files should be manually removed.

## 8 Searching of the EAMATE data

The following section outlines the high level data flow during the search process. The search engine runs in the background on the file server machine, waiting for a request from a client. The requests which can be handled by the search engine are as follows:

- Get Employer Header -- All Headers for the Given Year and EIN
- Get Employer Header -- By Sequence Number
- Single Query Request
- Blanket Request
- Get Employee Detail
- Browse Report
- Print Report
- Add Matches

Each of these requests are made via a remote procedure call (RPC call) by the client program to the search engine. This call alerts the server that a request is waiting, and that a query file is available on a shared hard disk. This query file, called query<#>.txt (where # is the user number), outlines the details of the query. It contains a structure of type USER\_QUERY with the query parameters for the request. The sections below will outline the steps taken to satisfy each of the types of requests.

The search engine is actually comprised of seven applications: "search\_header", "search\_single", "search\_blanket", "search\_detail", "search\_browse", "search\_print", and "search\_addmatch". These applications should be run in the background from the root directory of the account containing all of the data files on the file server. It is important that all seven of the applications are always running, and that only one copy of each application is running at a given time. To run an application, while in the home directory of the "ssa" account, at the prompt, type: <application name> & (for example, to run the search\_header application, at the prompt, type: search\_header &). It is VITAL that these applications are started while in the home directory of the "ssa" account, as this is where the index and duplicate posting files reside, and is the parent directory for all of the data sub directories.

### 8.1 Get Employer Header -- All Headers for Year & EIN (Application: "search\_header" )

When this request is received by the search engine, the user query file is opened and the year and EIN are extracted.

A file called header<#>.txt is created to hold the Employer Header Information structures. The 1991.employers.idx file is then searched for the particular EIN. If it is not found, then an error is displayed, and FAIL is returned to the calling program.



If the EIN is found, then, starting with the first report, the header information is read from the 1991.employers.txt file and written to the header<#>.txt file. This is continued until all report headers for this EIN and year have been copied. The function then returns SUCCESS to the calling program.

## **8.2 Get Employer Header -- By Sequence Number (Application: "search\_header")**

When this request is received by the search engine the user query file is open and the year, EIN, and report sequence number are extracted.

A file called header<#>.txt is created to hold the Employer Header Information structure. The 1991.employers.idx file is then searched for the particular EIN and report sequence number combination.

If the data combination is not found, then an error is displayed and FAIL is returned to the calling program. If the EIN/Report Sequence Number combination is found, then the header information of this report is read from the 1991.employers.txt file and written to the header<#>.txt file. The function then returns SUCCESS to the calling program.

## **8.3 Single Query Request (Application: "search\_single")**

When this request is received by the search engine, the user query file is opened and the year, EIN, first and last name, and SSN are extracted. The search statistics structure (struct TEST\_DATA) is initialized to gather information regarding the search. It is filled with the relevant search parameters available at this time (User Number, Year, EIN, First name, Last name, SSN).

A file, browse<#>.txt, is created to contain the browse information for the ranked matches. The header<#>.txt file is created to contain the first Employer Header Information structure for the year and EIN. The listoff<#>.txt file is created to store the ranked list of offsets for the search. The search<#>.txt file is opened to be updated with the search statistics for this search.

The 1991.employers.idx file is then searched looking for the year and EIN combination.

If it is not found, an error is displayed and FAIL is returned to the calling program. If the EIN is found, then the first Employer Header Information structure is copied to the header<#>.txt file, and the search process is continued.

The search name is formed from the first and last name passed from the client. This consists of the first name followed by the first initial followed by the last name. This parameter is added to the search statistics structure. Clock timers that measure the elapsed and CPU time are initialized, and the actual search begins.

This is done by parsing the name and the social security number into n-grams. Overlapping digrams are used for the name, and the logical breaks in the SSN are used to break it. For example, "JOE J BROWN", "212 88 3587" would be parsed as "JO", "OE", "J ", "BR", "RO", "OW", "WN", "212", "88", "3587".

Each one of the n-grams is searched in the btree, and a duplicate queue is returned for each n-gram. The duplicate queue consists of a list of browse record offsets in descending order. Each of these offsets point to a record which contains the n-gram.

Once all of the duplicate queues have been returned, the queues are merged to arrive at a rank for each record. The rank is based upon the number of n-grams in common between the record and the user query, and the lengths of the record and user query.

This ranking is done by a linear merge of the queues, and then by normalizing this raw score by the use of the Dice Coefficient. Records which rank below a certain score are pruned, and the remaining records are hashed into an array of linked lists, using their rank to determine to which linked list to add them.

The lists are then traversed from highest rank to lowest rank, writing each browse offset to the listoff<#>.txt file. This produces the ranked list of offsets to record matches. A subset of this list is then translated into browse information, and the ranked browse information is written to the browse<#>.txt file.

The test information structure is then filled with the remaining values (Number of n-grams searched, Number of record matches, Number of records pruned, Elapsed time, CPU time, Write time, and Memory allocated). If an error occurred in the search process, the nummatch field of the search statistics structure is set to -1. The search statistics structure is written to the search<#>.txt file.

The client is returned a value of SUCCESS. FAIL is only returned during the search process if the search engine encounters a memory error or a read/write error.

#### **8.4 Blanket Request (Application: "search\_blanket")**

When this request is received by the search engine, the user query file is opened and the year, EIN and report sequence number are extracted.

A file called header<#>.txt is created to hold the employer header information structure. A file called blanket<#>.txt is created to hold the employee detail records. The 1991.employers.idx file is searched for the particular EIN and sequence number combination. If it is not found, FAIL is returned to the calling program, and an error message is displayed on the workstation screen.



If the EIN/sequence number combination is found, the employer header information structure for the specified sequence number is written to the header<#>.txt file. BLANKETFACTOR records are read from the beginning, middle, and end of the specified employer report. These records are printed to blanket<#>.txt. A return value of SUCCESS is then sent to the calling program.

### **8.5 Get Employee Detail (Application: "search\_detail")**

When this request is received by the search engine, the user query file is opened and the year, EIN, sequence number and detail offset are extracted.

A file called detail<#>.txt is created to hold the employee detail record. A file called header<#>.txt is created to hold the employer header information record.

The 1991.employers.idx file is searched for the year/EIN/sequence number combination. If it is not found, then FAIL is returned to the calling program. Otherwise, the detail file is opened and the employer detail record specified by the offset is read. If the offset does not point to an employee detail record, then FAIL is returned to the calling program. Otherwise, the employee detail record is printed to the detail<#>.txt file, and SUCCESS is returned to the calling program.

### **8.6 Browse report (Application: "search\_browse")**

When this request is received by the search engine, the user query file is opened, and the year, EIN, and offset data are extracted. In this case, the offset field contains the MRN at which to start the browse.

A file called browse<#>.txt is created which will hold the browse records for the user. A file called header<#>.txt is created to hold the employer header information structure.

The browse file is opened and searched for the MRN. If the MRN is found, the first BROWSENUM records are copied from the browse file to the browse<#>.txt file. The employer header corresponding to the browse records is read from the 1991.employers.text file, and printed to the header<#>.txt file. A result of SUCCESS is returned to the user.

### **8.7 Print Report (Application: "search\_print")**

When this request is received by the search engine, the user query file is opened, and the year, EIN, and report sequence number are extracted.

A file called print<#>.txt is created which will hold the formatted employer detail report before it is printed. A file called header<#>.txt is created to hold the employer header information structure.



The 1991.employers.idx file is searched for the year/EIN/sequence number combination. If it is not found, then FAIL is returned to the calling program. Otherwise, the detail file is opened, and the data in the file is formatted and printed to the print<#>.txt file. The print<#>.txt file is then spooled to the printer, and SUCCESS is returned to the calling program.

NOTE: The print report function will return an error if the number of employees in the file is greater than 5,000. This is to provide some security to guard against huge files being printed. If it is necessary for a larger file to be printed, there is an application called "sysadm\_print" which can be run by the system administrator to print the employer report. See section 7.4 for more information.

### **8.8 Add Matches (Application: "search\_addmatch")**

When this request is received by the search engine, the user query file is opened and the year, EIN, and offset information are extracted. In this case, the offset field contains the location in the list of offsets file (listoff<#>.txt) at which to start adding matches.

A file called browse<#>.txt is created to hold the additional matches. The listoff<#>.txt file is opened to obtain the ranked list of matches, and the browse data file for the specified report is opened to obtain browse information. If any of these files do not exist, or cannot be created, then FAIL is returned to the calling program. Otherwise, the browse data is pulled for the next set of matches in the listoff<#>.txt file and placed in the browse<#>.txt file, and SUCCESS is returned to the calling program.



## 9 Communication Issues

Communication between the search engine and the user interface occurs in several ways. These include the use of Network File System (NFS) mounted hard disks on the workstation, and the use of Remote Procedure Calls (RPC) from the workstation to the server.

The NFS mounts are used as a means of passing data between the server and the workstation. They consist of a hard drive that is physically attached to the SUN file server being remotely connected to the workstation via a network mount. This allows the workstation to access the hard drive as though it was physically attached. Query parameters for a data request are written by the workstation to this shared hard drive, and query results are written by the file server to the shared hard drive as well.

The RPC functions are executable procedures that are resident on the SUN workstation, and are registered by the server for use by networked machines. They are accessed by the remote machine in the same way that a local function call is made. There are a series of RPC functions which are registered by the server when it is started. These RPC functions are called by the workstation when a request for information is necessary. Prior to making an RPC call the workstation writes the query parameters to a known file (query<#>.txt) on the shared hard disk. The parameter passed to each RPC call is the user number associated with a workstation, and the RPC call will return a 1 upon success and a -1 upon failure. The list of RPC calls and associated information follows.

Type of Request	Function Name	RPC Registration Numbers (program, version, procedure)	Reference Section
Get Employer Header (All)	get_emplr_header	0x30000000, 1, 1	8.1
Get Employer Header (By Seq)	get_seq_header	0x36000000, 1, 1	8.2
Single Query Request	single_query	0x31000000, 1, 1	8.3
Blanket Request	get_blanket	0x32000000, 1, 1	8.4
Get Employee Detail	get_empl_detail	0x34000000, 1, 1	8.5
Browse Report	browse_report	0x35000000, 1, 1	8.6
Print Report	print_report	0x33000000, 1, 1	8.7
Add Matches	add_matches	0x37000000, 1, 1	8.8

Table 9-1. Listing of RPC Functions





## 10 Usability

Myers and Rosson reported in 1992 that in recent years, user interface code has accounted for about half (on average) of developed software. From this observation, it is apparent that the user interface consumes a significant portion of design and development resources. Therefore, it is important that sound practices and guidelines be applied to the user interface during the design and development process to ensure the creation of an easy-to-use, easy-to-learn, successful interface. Effective application of usability engineering techniques plays an important role in achieving this goal.

Although the concept of usability may be seen as a part of overall system acceptability, according to Nielsen (1993) usability is specifically concerned with five basic components:

- Learnability - Is the system easy to learn?
- Efficiency - Can the user make efficient use of the system, becoming more productive while operating the system?
- Memorability - Is the system easy to remember in terms of task performance and navigation of the system?
- Errors - Is the system designed so that user errors and system errors are minimized and recoverable?
- Satisfaction - Do the users like the system?

The components of usability relate to all human-computer interaction aspects of a system therefore the software user interface is a prime candidate for the application of usability engineering design principles. While documentation, installation procedures and maintenance issues are important parts of a system and are included in the scope of human-computer interaction, they are not addressed here. For the purposes of this document, the term usability is used as it pertains to the EAMATE software application prototype with special emphasis on the user interface portion.

### 10.1 Incorporating the Usability Engineering Lifecycle

The usability engineering lifecycle outlined by Nielsen (1993) served as a baseline model for implementing usability concepts. With some variation and customization, the lifecycle was integrated into the iterative prototype software development process. This occurred in variable and shifting stages -- with some actions taking place concurrently, and other events happening in succession -- but with options to return to previous phases. The process used to develop the EAMATE User Interface Prototype does not follow a strict linear methodology as outlined in traditional software development techniques and neither does the usability portion that has been integrated into this process. Several of the usability stages overlapped with general system analysis functions and the iterative design process of the prototype. Usability stages discussed here and in following sections include user interviews and task analysis, parallel design, participatory design, user testing, application of general guidelines and heuristic evaluation, empirical testing, and subjective assessment.

As previously mentioned, NIST computer scientists conducted several information gathering sessions, with regular users as well as management users. The current scouting process was observed and SSA personnel demonstrated how the EAMATE files fit into the overall investigatory function. Presently, some of the investigation steps are automated and some are not. The automated portions are handled via a terminal connected to a mainframe. All paper, film and electronic information pertaining to the scouting process was examined by NIST personnel as part of task evaluation. Besides task analysis, it was also determined after the interviews, that most of the scouts had no PC experience, hardware as well as software. User knowledge/experience and task analysis provided preliminary design criteria.

Following initial information gathering, NIST computer scientists employed parallel design, comparing interpretations and deciding on a basic interface design utilizing shortcuts (see shortcuts section below). After initial design decisions, participatory design became the major design method utilized. Several user testing sessions along with a scouting consultant facilitated participatory design and iterative development.

Three user testing sessions were conducted to aid design and development. During the first two sessions, a skeleton interface using test data was displayed on a PC and groups of three to five scouts interacted with the interface and the designers for six 1-hour periods. This is a variation of the constructive-interaction method (Nielsen, 1993) which involves only two users instead of groups of users. The method worked well, most of the group discussions were enthusiastic and animated. Many of the participants expressed pleasure and satisfaction at being included in the design process. The majority of the scouts comments and suggestions were incorporated into the user interface prototype and a second session of user testing (involving the same individuals and format) was conducted, again using the group constructive-interaction method. One example of the advantage of participatory and user-centered design involves the format of the user interface in SVGA vs. VGA mode. Originally, NIST computer scientists designed the prototype for a VGA monitor so that it would be more readable. However, during user testing, the scouts identified more information than was physically possible to display in this mode, as important in determining the identification and verification of a target individual. Consequently, NIST personnel re-designed the interface for a SVGA monitor to accommodate more information on the screen. Additionally, this helped the scouts understand part of the re-engineering process concerning the separation of the data into browse data and detail data. The users helped identify and determine the contents of the browse data thereby acquiring knowledge about the system even before its full implementation. The change from VGA mode to SVGA mode was made before the second test and the scouts expressed satisfaction with the change.

NIST computer scientists supervised a third user testing session in which straight-forward training was conducted. Trainers directed test participants in the major uses of the prototype and answered whatever questions were posed. The third session involved training a new set of scouts to use the fully implemented EAMATE prototype. Over a six-and-a-half-day period, with five sessions a day, approximately twenty scouts were trained to use the system. The format of the training and testing consisted of two adjacent workstations with one user and one trainer each. Average training time (time to independent navigation) was 1 hour. After learning to maneuver around the system, the scouts spent their time trying to "fool" the system by entering bogus



queries. The test participants performed at least twenty searches per hour and were able to locate the required data. Users also identified other usability problems such as focus issues, unusual responses to key strokes and confusing terminology. (These problems were corrected for the expanded prototype testing.) At the conclusion of each training session, the scout was asked to write a paragraph describing his/her training experience and opinion of the prototype. These comments are included in Appendix E and represent initial subjective user satisfaction.

## **10.2 Heuristic Evaluation as a Usability Inspection Method**

During the initial design phase and user testing sessions, NIST personnel addressed usability throughout the development process using 10 basic design principles as outlined by Nielsen (1993). These basic guidelines cover a broad range of attributes and may be applied to user interfaces in general. In addition to design criteria, the guidelines were also used as the basis for heuristic evaluation, which occurred as part of the iterative design process. NIST computer scientists performed heuristic evaluation as a usability inspection method during user interface design and by observing test users during the early stages of development. Each of the basic heuristic usability characteristics is described below in the context of the development of the EAMATE prototype and heuristic evaluation examples are included where appropriate.

- **Simple and Natural Dialogue**

After preliminary task analysis a baseline screen was designed using X's so the layout could be graphically visualized and analyzed. Gestalt principles in terms of grouping like objects are incorporated into the user interface and in terms of color use for helping the user identify object relationships through categorization, differentiation, and highlighting. The minimum amount of data needed by the users to perform their jobs was determined through user-centered task-oriented examination and incorporated into the user interface prototype. Scrolling concepts are utilized to closely mimic the current manual operation allowing for a smoother transition between the current system and the prototype. Pushbutton choices on the main screen are organized in terms of tasks performed by the user and data is organized in terms of what the user needs and at what time the user needs it. Throughout the design phase and the concurrent heuristic evaluation, every effort was made to create a simple natural dialogue that was easy for the user to understand and to navigate.

- **Speak the User's Language**

Task and environment terminology are incorporated into the user interface where applicable. As part of the re-engineering process some new terms were introduced such as browse data and detail data but the users quickly adapted to these terms when they realized that separation of the data into these two types allowed them to perform their tasks in a more efficient manner. As one example of heuristic evaluation, during early design of the user interface prototype NIST personnel used the term "header" to indicate employer information. In technical terms this

makes sense to computer scientists since this was information located at the beginning of the employer report and pertained to the entire report much like a file header. However, NIST personnel observed that the term "header" was confusing to test users and changed the label to "employer" information. Consequently, users expressed satisfaction with the change and remarked that the terminology made a difference in their ability to understand labels in the interface.

Another aspect of speaking the user's language involves mappings. In order to map the manual task to the prototype, NIST personnel again made use of task analysis results. Differentiation of tasks aided in the design of the main screen and ultimately led to the five pushbutton choices. Observation of work flow provided the basis for presentation ordering in the user interface prototype. Identification of related and common tasks led to the inclusion of detail data access across the information display screens. An illustration of heuristic evaluation involving mappings is the "potential blanket" option. Originally, this option was one of the five choices on the main screen, but after early test sessions, test participants revealed that the identification of a potential blanket adjustment occurred while in the process of a single query. As a result, NIST personnel adjusted the interface to reflect this event and the "potential blanket choice" became an option on the single query information display screen instead of a choice on the main screen.

- **Minimize User Memory Load**

Every operation contained in the user interface prototype may be made by pressing a pushbutton or clicking the mouse except data entry operations, which are necessary to communicate an initial data request. Since the volume of information is so huge it is not possible to present the user with a set of data (in varying combinations) to choose from in terms of forming a query. However, once past the query stage all other operations are presented in a manner that the user may choose from instead of remembering specific data to input. Clear, easy-to-follow instructions are incorporated throughout the application to ease user memory load. Data is accessed in the same manner across the different screens further minimizing user memory load.

- **Consistency**

As previously mentioned in other parts of this document, consistency is maintained throughout the user interface prototype through data organization, format, location, access methods, and operation functionality.



- **Feedback**

Feedback is provided through use of icons (e.g., the hourglass icon indicating a wait state), message beeps, and message boxes. Enabling and disabling of windows also provides a method of feedback in conjunction with the use of color, which indicates active and inactive windows. Response time to a user's query is determined by a number of factors, such as the commonality of the name, the degree of error in the search query, and the amount of data being searched. Detailed discussion of response/search times as it pertains to the search engine is discussed in section 8. Full response time, along with a variety of other important application performance statistics, will be measured during the planned expanded prototype and reported in a subsequent publication.

- **Clearly Marked Exits**

Exit options are present on every major screen of the user interface prototype to provide control. Users may exit the application from the main screen simply by selecting the "Exit Program" pushbutton. On the query screens, users have the option to "Cancel" the current query and on the information display screens users may "Close" out the current display. To remove a detail screen, users simply "Continue" in order to be returned to the information display screen. Clearly marked exits allow full exploitation of the application by offering easy methods for canceling operations, closing out completed operations, and by removing the "fear factor" of being lost in the software with no visible means of escaping.

- **Shortcuts**

Shortcuts include accelerators, function keys, pushbuttons, command histories, and generally any method that allows users, at their discretion, to perform an operation in a quicker manner than currently incorporated application operations permit. As an initial design decision, NIST personnel opted for "shortcuts" as the preferred and major method of interaction, not just as options, in the EAMATE user interface prototype. After observing the user population and determining basic process functionality, NIST computer scientists chose pushbuttons and mouse operations as predominant techniques for user navigation of the software. However, as fundamental interface design dictates, and basic heuristic evaluation revealed, inclusion of keyboard operations as well as mouse operations was necessary for those users who prefer not to use the mouse. But, the keyboard operations still function as shortcuts in terms of "pushing buttons" or substitutions for mouse clicks as related to application navigation.

- **Good Error Messages**

Error messages are included in a variety of locations throughout the user interface software. Simple, concise messages indicate data entry rules and requirements. Data access errors are communicated to the user through use of icons and special dialog boxes. Error messages are



worded in a non-combative/non-judgmental tone so as not to put any unnecessary anxiety upon the user and to further enhance user comfortableness. Error messages alert the user to a specific problem where possible, i.e. the message, "STOP, System Unable to Open File Query#.txt, See System Administrator" appears when the file handle returns NULL upon attempted file access. If necessary, as illustrated, error messages also tell the user when to contact the system administrator for help. Specific, understandable, error messages permit the user to precisely identify problems and inform the system administrator. In addition, network communication and system errors are recorded in a file to facilitate troubleshooting by appropriate personnel. A complete listing of error dialog boxes, and error messages and their meanings may be found in Appendix C.

- **Prevent Errors**

It is preferable to design a system such that errors are prevented before they have a chance to occur. To prevent data entry errors from being communicated to the search engine, message beeps and message boxes are incorporated into the query screens and guide users in formatting correct queries. Additionally, text input is automatically formatted as uppercase removing another layer of possible data entry errors regarding uppercase and lowercase input or any combination thereof. As another error prevention technique, query fields are enabled with automatic mode which determines whether characters will be replaced or inserted. This attribute relieves the user of having to manually adjust modes and also helps avoid errors caused by not knowing or realizing the current mode of the data entry field. Standard prevention measures such as message beeps upon entry attempt of an illegal character or an illegal mouse operation are also incorporated into the EAMATE user interface prototype.

- **Help and Documentation**

Even though the EAMATE application is a prototype and not a full-fledged system, a certain amount of help and documentation has been included and prepared. Currently, on-line help and documentation are not available. However, basic "Help" is provided in terms of screen instructions, error prevention methods, and error reporting techniques as described in Appendix C; user documentation is provided in Appendix A which contains step-by-step operating instructions, a visual manual and a set of data entry rules. Technical documentation is provided via this publication.

### **10.3 Application Performance Measurements and Usage Statistics**

After usability inspection and concurrent preliminary user testing, NIST personnel composed a list of application performance measurements and usage statistics to be gathered during the expanded prototype. Resulting values will provide a picture about how the application is being used, its efficiency as compared to the current system, and a series of quantitative measures, e.g. response time. User satisfaction will be measured by a questionnaire which is described later in

this section. Preliminary user satisfaction was measured during early testing and development through user comments and feedback. Appendix E contains a list of early test participants' comments. Below is a list of the planned application measurements and statistics with a brief description of each. Following the list is a description of how these measurements/statistics are incorporated into the user interface software and how associated functions and files are used to accomplish the gathering and subsequent accumulation and reporting process.

## Statistics Structures

Application performance measurements and usage statistics are stored in two user-defined C structures implemented as an internal structure of type `in_stat` and an external structure of type `ex_stat`. Each structure type is implemented as a set of numeric types including integers, doubles, and some time types (in seconds). Type `char` is also used but as a numeric value not an alpha value. The structure named "current" of type `in_stat` is used to store information about the current operation (either a single-query or a browse) and is only visible to the application. The structure named "aggregate" of type `ex_stat` is used to hold cumulative information, is updated continually, and is visible outside the application via a file. The structure `current` and the structure `aggregate` contain the following members:

### current structure members:

<code>interrupted</code>	<code>//yes=1, no=0, were there any interruptions //during this case?</code>
<code>resolved</code>	<code>//yes=1, no=0, was this case resolved?</code>
<code>outlier</code>	<code>//yes=1, no=0, did this case take longer than //a 1/2 hour to complete?</code>
<code>snet_time</code>	<code>//net time of a single query case from start to finish</code>
<code>qnet_time</code>	<code>//net time of a single query case from display //of matches to finish</code>
<code>bnet_time</code>	<code>//net time of a browse case from start to finish</code>
<code>add_match</code>	<code>//count of how many times additional matches //is pressed during a single query case</code>
<code>start_single</code>	<code>//start time of a single query case</code>
<code>start_qinfo</code>	<code>//start time of display of matches during a single query</code>
<code>stop_single</code>	<code>//stop time of a single query case</code>
<code>start_browse</code>	<code>//start time of a browse case</code>
<code>stop_browse</code>	<code>//stop time of a browse case</code>

The members of the aggregate structure are divided into three types: totals used for calculating averages and percentages, averages and percentages, and usage totals per selected operation. Structure members are listed below in terms of the three sections. (NOTE: Cases refer to single query cases unless otherwise stated.)

### aggregate structure members:



### Type 1 - Totals Used for Calculating Averages and Percentages

br_tot_time	//running total of net browse time
rs_tot_time	//running total of net single query time of resolved //uninterrupted cases
us_tot_time	//running total of net single query time of unresolved //uninterrupted cases
rsi_tot_time	//running total of net single query time of resolved //interrupted cases
usi_tot_time	//running total of net single query time of unresolved //interrupted cases
rq_tot_time	//running total of net single query time (from display //of matches) of resolved uninterrupted cases
uq_tot_time	//running total of net single query time (from display //of matches) of unresolved uninterrupted cases
rqi_tot_time	//running total of net single query time (from display //of matches) of resolved interrupted cases
uqi_tot_time	//running total of net single query time (from display //of matches) of unresolved interrupted cases
rtot_non_interrupt	//running total of resolved uninterrupted cases
utot_non_interrupt	//running total of unresolved uninterrupted cases
tot_non_interrupt	//running total of uninterrupted cases
rtot_interrupt	//running total of resolved interrupted cases
utot_interrupt	//running total of unresolved interrupted cases
tot_interrupt	//running total of interrupted cases
tot_browse	//running total of completed browse cases
tot_cases	//running total of cases
tot_non_cases	//running total of incomplete cases
tot_resolved	//running total of resolved cases
tot_unresolved	//running total of unresolved cases
tot_outliers	//running total of number of case with a net time > 1/2 //hour
tot_add_records	//running total of number of times "add records" is //pushed
tot_add_matches	//running total of number of times "additional matches" //is pushed
rtot_add_match	//running total of number of times "additional matches" //is pressed for resolved cases
rtot_add_match0	//running total of number of resolved cases in which //"additional matches" was not pressed
rtot_add_match1	//running total of number of resolved cases in which //"additional matches" was pressed once
rtot_add_match2	//running total of number of resolved cases in which //"additional matches" was pressed twice
rtot_add_match3	//running total of number of resolved cases in which



	// "additional matches" was pressed three times
rtot_add_match4_plus	// running total of number of resolved cases in which
	// "additional matches" was pressed four or more times
utot_add_match	// running total of number of times "additional matches"
	// is pressed for unresolved cases
utot_add_match0	// running total of number of unresolved cases in which
	// "additional matches" was not pressed
utot_add_match1	// running total of number of unresolved cases in which
	// "additional matches" was pressed once
utot_add_match2	// running total of number of unresolved cases in which
	// "additional matches" was pressed twice
utot_add_match3	// running total of number of unresolved cases in which
	// "additional matches" was pressed three times
utot_add_match4_plus	// running total of number of unresolved cases in which
	// "additional matches" was pressed four or more times

## Type 2 - Averages and Percentages

avg_br_time	// average net browse time
avg_add_record	// average number of times "add records" is pressed
	// during a browse case
rs_avg_time	// average net time per resolved uninterrupted case
rsi_avg_time	// average net time per resolved interrupted case
us_avg_time	// average net time per unresolved uninterrupted case
usi_avg_time	// average net time per unresolved interrupted case
rq_avg_time	// average net time per resolved uninterrupted case
	// (from display of matches)
rqi_avg_time	// average net time per resolved interrupted case
	// (from display of matches)
uq_avg_time	// average net time per unresolved uninterrupted case
	// (from display of matches)
uqi_avg_time	// average net time per unresolved interrupted case
	// (from display of matches)
ravg_add_match	// average number of additional matches per resolved case
uavg_add_match	// average number of additional matches per unresolved
	// case
avg_add_match	// average number of additional matches per case
rper_add_match0	// percentage of resolved cases in which no additional
	// matches were requested
rper_add_match1	// percentage of resolved cases in which additional matches
	// was requested once
rper_add_match2	// percentage of resolved cases in which additional matches
	// was requested twice
rper_add_match3	// percentage of resolved cases in which additional matches
	// was requested three times
rper_add_match4_plus	// percentage of resolved cases in which additional matches

	//was requested four or more times
uper_add_match0	//percentage of unresolved cases in which no additional
	//matches were requested
uper_add_match1	//percentage of unresolved cases in which additional
	//matches was requested once
uper_add_match2	//percentage of unresolved cases in which additional
	//matches was requested twice
uper_add_match3	//percentage of unresolved cases in which additional
	// matches was requested three times
uper_add_match4_plus	//percentage of unresolved cases in which additional
	// matches was requested four or more times

### Type 3 - Usage Totals Per Selected Operation

tot_single_query	//running total of single query selection
tot_browse_report	//running total of browse report selection
tot_print_report	//running total of print report selection
tot_blanket	//running total of potential blanket selection
tot_diff_report	//running total of select different report operation
tot_qp	//running total of display current parameters operation
tot_er_detail	//running total of employer detail selection
tot_ee_detail	//running total of employee detail selection
tot_final	//running total of report totals selection
tot_pr_er_detail	//running total of print employer detail selection
tot_pr_ee_detail	//running total of print employee detail selection
tot_ee_detail_printed	//running total of number of employee details printed
tot_pr_final	//running total of print final totals selection
tot_pr_blanket	//running total of print blanket selection
tot_pr_report	//running total of OK selection in print report data entry
	//dialog

## **Statistics Functions**

The code module stat.c contains five functions that operate on the gathered performance measurements and usage statistics. The five functions have already been listed in section 4.5 under the heading Function Organization but are listed here again for ease of reading and subject relativity. For a detailed code listing of stat.c see Appendix D - Listing of the Code.

### Statistical Gathering and Calculation Functions

- Fill\_Aggregate() - fills and calculates aggregate statistics
- Fill\_Current() - calculates times for current statistics
- Init\_Aggregate() - initializes aggregate statistics structure
- Init\_Current() - initializes current statistics structure
- Write\_Aggregate() - writes aggregate statistics to a file



## Statistics Implementation

The statistics are gathered as events occur, operated on as necessary and recorded in a file for further use. Basically, when the application is opened, the previous aggregate statistics are read from a local binary file stat#.fil (# refers to the user number) and the aggregate structure is filled. When a single query or browse report operation is begun, the current structure is initialized to all zeroes. At the conclusion of a single query or browse report operation, current calculations are made and recorded, then added to the aggregate totals. Aggregate totals are then updated and recorded in the file stat#.fil. At the conclusion of a single query operation, users are asked to respond to two on-line questions: 1) Was the case resolved? 2) Did any interruptions occur? These questions will allow more detailed analysis at the conclusion of the expanded prototype and will be discussed more fully in a subsequent publication regarding analysis of the measurements/statistics being gathered.

In addition to being recorded upon completion of single query and browse operations, aggregate totals are also recorded upon cancellation of a single query, completion of a potential blanket operation, successful completion of the print report data entry screen, and upon exiting the application. Periodic updating and appropriate placement of update functions ensures a reliable capture process.

SSA/NIST plan to conduct testing on the EAMATE application prototype for a period of six months on two separate functional units each consisting of approximately 15 participants with two persons per workstation. In order to observe trends as well as obtain overall statistical data, NIST personnel recommend that statistics be monitored on a monthly basis per both test groups of users during the expanded prototype. For detailed instructions on setting up the monitoring process, see Appendix B.

### 10.4 User Satisfaction

Besides quantitative performance measurements and usage statistics, it is also important to look at qualitative and subjective indicators. One tool used in determining the quality and effectiveness of an application is an instrument requiring user input and judgment. NIST personnel intend to administer a user satisfaction questionnaire one month into the expanded prototype and at the conclusion of the expanded prototype to gauge initial and long-term satisfaction rates and to determine if any adjustments and/or additions are desired. The user satisfaction questionnaire is currently under development.

Design of the questionnaire will be guided by examination of existing user satisfaction instruments (Chin et al, 1988), current literature in the field of human-computer interaction, and standard questionnaire development guidelines. NIST personnel will tailor the questionnaire as related to interface content and the existing SSA user community. Terminology will be geared towards the test subjects' vocabulary. Scaling will be incorporated to aid in reliability (Chin et. al., 1988) and an even-numbered scale will be used to prompt the evaluator to make choices



instead of choosing the mean for every question. While test users' comments were gathered during very early testing of the prototype, the questionnaire represents a more structured approach to compiling user satisfaction measurements.

## **10.5 Conclusions and Future Plans**

Preliminary results of integrating usability into the iterative prototyping process support the integration decision. During early training and testing, learning times (time it took user to perform independent navigation) ranged from 1/2 hour to 1-1/2 hours with an average time of 1 hour. After discussion with the test participants, it was determined that under the current manual system, SSA scouts processed roughly 8-10 cases a day. Following observation of test participants performing their investigative procedures using the EAMATE prototype application, NIST personnel conservatively estimated that 8-10 cases an hour could be processed. Additionally, both verbal and written user feedback indicated a high degree of user satisfaction (see Appendix E).

After the application performance measurements, usage statistics, and results of the user satisfaction questionnaire have been calculated and analyzed, NIST personnel will report the findings and relate them to overall system performance as well as to usability. NIST will also communicate results in terms of how the application is being utilized by the scouts, making sure to include a comparison/contrast between the two functional units in addition to identifying overall group trends.

## 11 References

- Adler, P. & Winograd, T., Eds. (1992). Usability. New York, New York: Oxford University Press.
- Angell, R., Freund, G. & Willett, P. (1983). Automatic spelling correction using a trigram similarity measure. *Information Processing and Management*, 19, 255-261.
- Borgman, C. & Siegfried, S. (1992). Getty's Synoname and Its Cousins: A survey of Applications of Personal Name-Matching Algorithms. *Journal of the American Society for Information Science*, 7, 459-476.
- Chin, J., Diehl, V., & Norman, K. (1988). Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface. *CHI '88 Conference Proceedings*, 213-218.
- Cormen, T., Leiserson, C. & Rivest, R. (1990). *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press.
- Damerau, F. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7, 171-176.
- Davidson, L. (1962). Retrieval of misspelled names in an airline's passenger record system. *Communications of the ACM*, 5, 169-171.
- Freund, G. & Willett, P. (1982). On-line Identification of Word Variants and Arbitrary Truncation Searching using a String Similarity Measure. *Information Technology: Research and Development*, 1, 177-187.
- Gadd, T. (1988). 'Fishing fore werds': phonetic retrieval of written text in information systems. *Program*, 22, 222-237.
- Hall, P. & Dowling, G. (1980). Approximate String Matching. *Computing Surveys*, 4, 381-402.
- Harman, D. & Candela, G. (1990). Retrieving Records from a Gigabyte of Text on a Minicomputer Using Statistical Ranking. *Journal of the American Society for Information Science*, 8, 581-589.
- Knuth, D. (1973). *The Art of Computer Programming: Sorting and Searching*. Reading, Massachusetts: Addison-Wesley.
- Mack, R. & Nielsen, J. (1993). Usability Inspection Methods - Report on a Workshop Held at CHI '92. *SIGCHI Bulletin*, 25, 1, 28-33.

- Maguire, M. & Dillon, A. (1993). Usability Measurement - Its Practical Value to the Computer Industry. Human Factors in Computing Systems - INTERCHI '93 Conference Proceedings, 145-148.
- Mayhew, D. (1992). Principles and Guidelines in Software User Interface Design. Englewood Cliffs, New Jersey: Prentice Hall.
- Nielsen, J. (1993). Usability Engineering. Boston, Massachusetts: Academic Press, Inc.
- Nielsen, J. & Molich R. (1990). Heuristic Evaluation of User Interfaces, CHI '90 Conference Proceedings, 249-256.
- Perry, S. & Willett, P. (1983). A review of the use of inverted files for best match searching in information retrieval systems. Journal of Information Science, 6, 59-66.
- Petzold, C. (1992). Programming Windows 3.1 - 3rd Edition. Redmond, Washington: Microsoft Press.
- Pollock, J. (1980). SPEEDCOP: Task A.1 - Quantification. Chemical Abstracts Service Internal Report.
- Pollock, J. (1981). SPEEDCOP: Task B.2 - Automatic Correction of misspellings. Chemical Abstracts Service Internal Report.
- Robertson, A. & Willett, P. (1992). Searching for Historical Word-Forms in a Database of 17th-Century English Text Using Spelling-Correction Methods. Proceedings of the 15th annual International SIGIR '92, 256-265.
- Rogers, H. & Willett, P. (1991). Searching for historical word forms in text databases using spelling correction methods: reverse error and phonetic coding methods. Journal of Documentation, 47, 333-353.
- Russell, R. (1918). United States patent 1261167. Washington, United States Patent Office.
- Russell, R. (1922). United States patent 1435663. Washington, United States Patent Office.
- Schulmeyer, G. & McManus, J. (1992). Handbook of Software Quality Assurance, 2nd Edition. New York, New York: Van Nostrand Reinhold.
- Wagner, R. & Fisher, M. (1974). The string-to-string correction problem. Journal of the ACM, 21, 168-173.





